

Name: - Meet Soni

ID NO: - 21EL035

Division: - 04

Year: - 2023-24

Subject: - Digital System Design (3EL42)

Branch: - Electronics (EL)

Assignment: - 2

Q-1: -Design 4-bit Ripple Carry Adder with the help of 1-bit adder

Verilog Code:-

```
1 module full_adder(in0, in1, cin, out, cout);
2     input in0, in1, cin;
3     output out, cout;
4
5     assign out = in0 ^ in1 ^ cin;
6     assign cout = ((in0 ^ in1) & cin) | (in0 & in1);
7 endmodule
8 module ripple_carry_adder(in0, in1, out, cout);
9     input [3:0] in0;
10    input [3:0] in1;
11    output [3:0] out;
12    output cout;
13
14    wire c1, c2, c3;
15    full_adder fa0(in0[0], in1[0], 0, out[0], c1);
16    full_adder fa1(in0[1], in1[1], c1, out[1], c2);
17    full_adder fa2(in0[2], in1[2], c2, out[2], c3);
18    full_adder fa3(in0[3], in1[3], c3, out[3], cout);
19
20 endmodule
```

Test Bench:-

```
module ripple_carry_adder_tb;                                SV/Verilog Tester

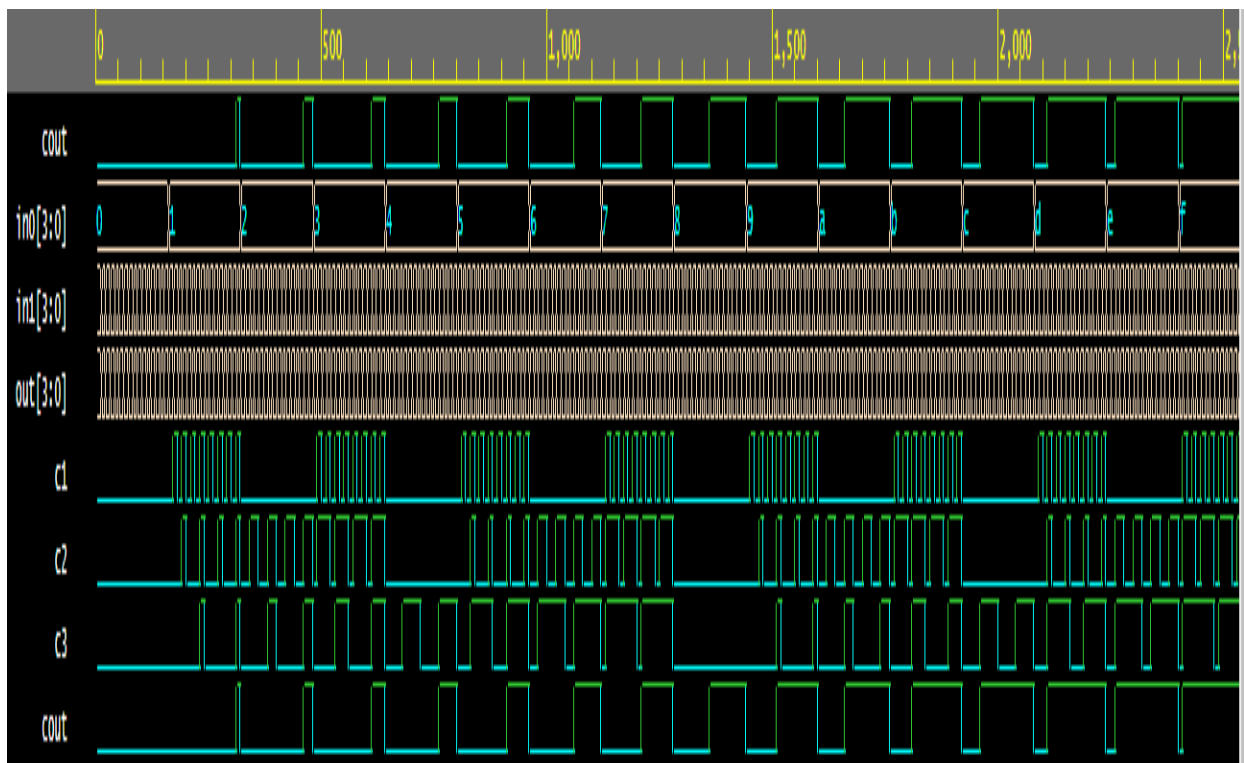
    reg [3:0] in0;
    reg [3:0] in1;
    wire [3:0] out;
    wire cout;

    ripple_carry_adder rca(.in0(in0), .in1(in1), .out(out),
    .cout(cout));

    initial begin
        in0 = 4'b0000; in1 = 4'b0000; #10
        in0 = 4'b0000; in1 = 4'b0001; #10
        in0 = 4'b0000; in1 = 4'b0010; #10
        in0 = 4'b0000; in1 = 4'b0011; #10
        in0 = 4'b0001; in1 = 4'b0000; #10
        in0 = 4'b0010; in1 = 4'b0000; #10
        in0 = 4'b0010; in1 = 4'b0001; #10
        in0 = 4'b0010; in1 = 4'b0010; #10
        in0 = 4'b0010; in1 = 4'b0011; #10
        in0 = 4'b0011; in1 = 4'b0000; #10
        in0 = 4'b0011; in1 = 4'b0001; #10
        in0 = 4'b0011; in1 = 4'b0010; #10
        in0 = 4'b0011; in1 = 4'b0011; #10
        in0 = 4'b0100; in1 = 4'b0000; #10
        in0 = 4'b0100; in1 = 4'b0001; #10
        in0 = 4'b0100; in1 = 4'b0010; #10
        in0 = 4'b0100; in1 = 4'b0011; #10
        in0 = 4'b0100; in1 = 4'b0100; #10
        end

    initial begin
        $dumpfile("ripple-carry-adder.vcd");
        $dumpvars(0, ripple_carry_adder_tb);
        $monitor($time, ": %b + %b = %b, %b", in0, in1, out,
    cout);
    end
endmodule
```

Output:-



Q-2: - Design D-flipflop and reuse it to implement 4- bit Johnson Counter

Verilog Code:-

```
module dflip_flop(out,d,reset,clk);
input d,clk,reset;
output out;

reg q;

always @(posedge reset or negedge clk)
begin

if(reset)
out=1'b0;
else
begin
out=d;
end
end
endmodule
module johnson_counter(q,clk,reset,q0);
input clk,reset,q0;
output q;
wire q1,q2,q3;
assign q0=~out;
assign dflip_flop d1(q1,clk,reset,q0);
assign dflip_flop d2(q2,clk,reset,q1);
assign dflip_flop d3(q3,clk,reset,q2);
assign dflip_flop d4(out,clk,reset,q3);
endmodule
```

Test Bench:-

```
module jc_tb;
reg clk,reset;
wire out;

johnson_counter dut (.out(out), .reset(reset), .clk(clk));

always
#5 clk =~clk;

initial begin
reset=1'b1; clk=1'b0;
#20 reset= 1'b0;
end

initial
begin
$monitor( $time, " clk=%b, out= %b, reset=%b",
clk,out,reset);
#105 $stop;
end
endmodule
```

SV/Verilog Testbench

Output:-

```
0  clk=0, out= xxxx, reset=1
5  clk=1, out= 0000, reset=1
10 clk=0, out= 0000, reset=1
15 clk=1, out= 0000, reset=1
20 clk=0, out= 0000, reset=0
25 clk=1, out= 0001, reset=0
30 clk=0, out= 0001, reset=0
35 clk=1, out= 0011, reset=0
40 clk=0, out= 0011, reset=0
45 clk=1, out= 0111, reset=0
50 clk=0, out= 0111, reset=0
55 clk=1, out= 1111, reset=0
60 clk=0, out= 1111, reset=0
65 clk=1, out= 1110, reset=0
70 clk=0, out= 1110, reset=0
75 clk=1, out= 1100, reset=0
80 clk=0, out= 1100, reset=0
85 clk=1, out= 1000, reset=0
90 clk=0, out= 1000, reset=0
95 clk=1, out= 0000, reset=0
100 clk=0, out= 0000, reset=0
```

Q-3: - Reuse 2:1 Mux code to implement 8:1 Mux

Verilog Code:-

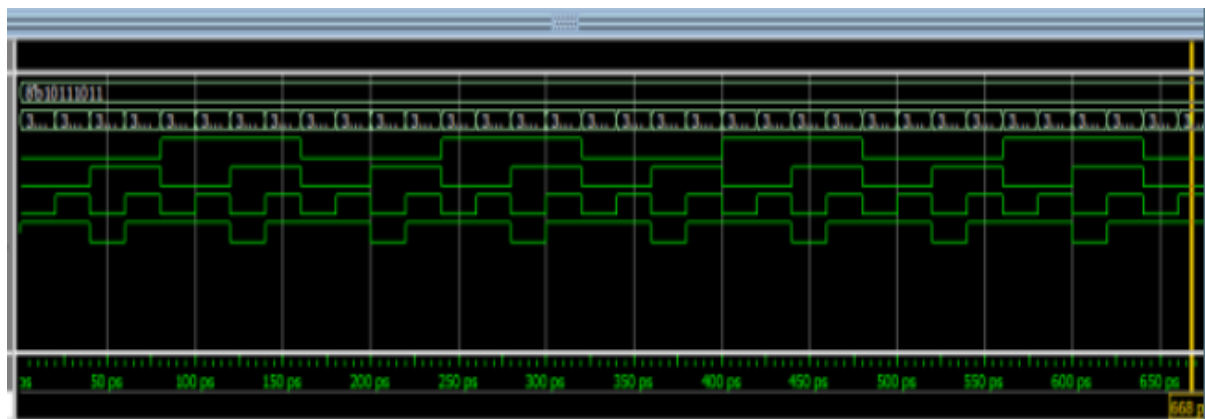
```
module mux8_1(i,s,y);
input [7:0]i;
input [2:0]s;
output y;
wire w1,w2,w3,w4,w5,w6;
mux2_1 m1(.i0(i[0]),.i1(i[1]),.s(s[0]),.y(w1));
mux2_1 m2(.i0(i[2]),.i1(i[3]),.s(s[0]),.y(w2));
mux2_1 m3(.i0(i[4]),.i1(i[5]),.s(s[0]),.y(w3));
mux2_1 m4(.i0(i[6]),.i1(i[7]),.s(s[0]),.y(w4));
mux2_1 m5(.i0(w1),.i1(w2),.s(s[1]),.y(w5));
mux2_1 m6(.i0(w3),.i1(w4),.s(s[1]),.y(w6));
mux2_1 m7(.i0(w5),.i1(w6),.s(s[2]),.y(y));
endmodule

module mux2_1(input i0,i1,s,output y);
assign y=(s==0)? i0:i1;
endmodule
```

Test Bench:-

```
module tb;
reg [7:0]i;
reg [2:0]s;
wire y;
mux8_1 DUT(i,s,y);
initial
begin
#15 i=8'b00011100;s=3'b011;
#15 i=8'b00110000;s=3'b000;
#15 i=8'b00111000;s=3'b011;
#15 i=8'b01110000;s=3'b100;
#15 i=8'b00110000;s=3'b101;
#15 i=8'b00001110;s=3'b001;
#15 i=8'b11110010;s=3'b010;
#15 i=8'b00011111;s=3'b110;
#15 i=8'b11000000;s=3'b000;
#15 i=8'b00010000;s=3'b111;
#15 i=8'b00000011;s=3'b011;
$dumpfile("mux8_1.vcd");
  $dumpvars();
end
endmodule
```

Output:-



Q-4: - Design a Full Subtractor with Gate Level Modelling Style. (Use primitive gates)

Verilog Code:-

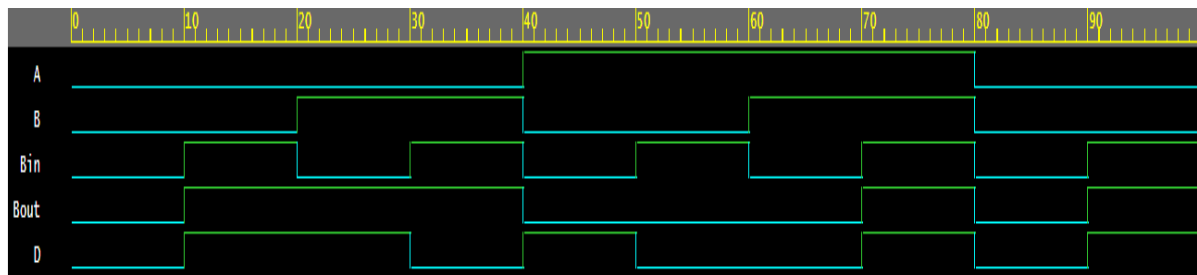
```
module full_sub(Bout,D,A,B,Bin);
output Bout,D;
input A,B,Bin;
wire w1,w4,w5,w6;
  xor (D,A,B,Bin);
  not n1(w1,A);
  and a1(w4,w1,B);
  and a2(w5,w1,Bin);
  and a3(w6,B,Bin);
  or o1(Bout,w4,w5,w6);
endmodule
```

Test Bench:-

```
module tb;
  reg A,B,Bin;
  wire D,Bout;
  initial begin

    // Add stimulus here
    #100; A = 0;B = 0;Bin = 1;
    #100; A = 0;B = 1;Bin = 0;
    #100; A = 0;B = 1;Bin = 1;
    #100; A = 1;B = 0;Bin = 0;
    #100; A = 1;B = 0;Bin = 1;
    #100; A = 1;B = 1;Bin = 0;
    #100; A = 1;B = 1;Bin = 1;
  end
  $dumpfile("full_sub.vcd");
  $dumpvars();
endmodule
```

Output:-



Q-5: - Design a 2X4 decoder using gate level modelling

Verilog Code:-

```
module decoder24_gate(en,a,b,y);
    input en,a,b;
    output [3:0]y;
    wire enb,na,nb;
    not n0(enb,en);
    not n1(na,a);
    not n2(nb,b);

    nand n3(y[0],enb,na,nb);
    nand n4(y[1],enb,na,b);
    nand n5(y[2],enb,a,nb);
    nand n6(y[3],enb,a,b);

endmodule
```

Test Bench:-

```
module tb;
    reg a,b,en;
    wire [3:0]y;
    decoder24_gate dut(en,a,b,y);

    initial begin
        $monitor("en=%b a=%b b=%b y=%b",en,a,b,y);
        en=1;a=1'b1;b=1'b1;#5
        en=0;a=0;b=0;#5
        en=0;a=0;b=1;#5
        en=0;a=1;b=0;#5
        en=0;a=1;b=1;#5
        $finish;
    end
endmodule
```

Output:-

```
# KERNEL: en=0 a=x b=x y=xxxx
# KERNEL: en=1 a=0 b=0 y=1111
# KERNEL: en=1 a=0 b=1 y=1111
# KERNEL: en=1 a=1 b=0 y=1111
# KERNEL: en=1 a=1 b=1 y=1111
```

Q-6: - Design a 4x1 mux using operators. (Use data flow)

Verilog Code:-

```
module mux4_1(y, i0,i1,i2,i3,[1:0]sel);  
    input i0, i1, i2, i3;  
    input [1:0]sel;  
    output y;  
    assign out = sel[1]? (sel[0]?i3:i2) : (sel[0]?i1:i0);  
endmodule
```

Test Bench:-

```
module tb;  
    reg i0,i1,i2,i3,[1:0]sel;  
    wire y;  
    mux4_1 dut(i0,i1,i2,i3,[1:0]sel,y);  
  
    initial begin  
        $monitor("sel=%b ->i3=%b,i2=%b,i1=%b,i0=%b->y=%b"  
sel,i3,i2,i1,i0,y);  
        sel=2'b00;  
        i0=1;i1=0;i2=1;i3=0;  
        sel=2'b01;  
        i0=1;i1=0;i2=1;i3=0;  
        sel=2'b10;  
        i0=1;i1=0;i2=1;i3=0;  
        sel=2'b11;  
        i0=1;i1=0;i2=1;i3=0;  
        $finish;  
    end  
endmodule
```

Output:-

```
sel = 00 -> i3 = 0, i2 = 1 ,i1 = 0, i0 = 1 -> y = 1  
sel = 01 -> i3 = 0, i2 = 1 ,i1 = 0, i0 = 1 -> y = 0  
sel = 11 -> i3 = 0, i2 = 1 ,i1 = 0, i0 = 1 -> y = 0  
sel = 01 -> i3 = 0, i2 = 1 ,i1 = 0, i0 = 1 -> y = 0
```

Q-7: - Design a Full adder using half adder.

Verilog Code:-

```
module half_addr(input a, b, output s, c);
    assign s = a^b;
    assign c = a & b;
endmodule

module full_adder(input a, b, cin, output s_out, c_out);
    wire s, c0, c1;
    half_addr HA1 (a, b, s, c0);
    half_addr HA2 (s, cin, s_out, c1);

    assign c_out = c0 | c1;
endmodule
```

Test Bench:-

```
module tb_top;
    reg a, b, c;
    wire s, c_out;

    full_adder fa(a, b, c, s, c_out);

    initial begin
        $monitor("At time %0t: a=%b b=%b, cin=%b, sum=%b,
carry=%b", $time, a,b,c,s,c_out);
        a = 0; b = 0; c = 0; #1;
        a = 0; b = 0; c = 1; #1;
        a = 0; b = 1; c = 0; #1;
        a = 0; b = 1; c = 1; #1;
        a = 1; b = 0; c = 0; #1;
        a = 1; b = 0; c = 1; #1;
        a = 1; b = 1; c = 0; #1;
        a = 1; b = 1; c = 1;
    end
endmodule
```

SV/Verilog Testbench

Output:-

```
At time 0: a=0 b=0, cin=0, sum=0, carry=0
At time 1: a=0 b=0, cin=1, sum=1, carry=0
At time 2: a=0 b=1, cin=0, sum=1, carry=0
At time 3: a=0 b=1, cin=1, sum=0, carry=1
At time 4: a=1 b=0, cin=0, sum=1, carry=0
At time 5: a=1 b=0, cin=1, sum=0, carry=1
At time 6: a=1 b=1, cin=0, sum=0, carry=1
At time 7: a=1 b=1, cin=1, sum=1, carry=1
```