BIRLA VISHVAKARMA MAHAVIDYALAY
(AN AUTONOMOUS INSTITUTION)
ELECTRONICS ENGINNERING DEPARTMENT
A.Y. 2023-2024

DIGITAL SYSTEM DESIGN

ASSIGNMENT - 2

NAME: RAVIKUMAR BARAIYA

ID NO: 21EL009

SEMESTER: 5TH

BATCH: A – BATCH

BRANCH: ELECTRONICS ENGINEERING

Q1. Design 4-bit Ripple Carry Adder with the help of 1-bit adder.

Code for Test-bench:

```
module testbench;

  reg [3:0] X;
  reg [3:0]Y;
  reg Cin;
  wire [3:0] S;
  wire Cout;

  ripple_carry_adder dut(X,Y,Cin,S,Cout);

    initial begin
      $monitor("X=%b Y=%b Cin=%b S=%b Cout=%b",X,Y,Cin,S,Cout);

      X[3:0]=4'b0000; Y[3:0]=4'b0000; Cin=0;  #5
      X[3:0]=4'b0001; Y[3:0]=4'b0001; Cin=0; #5
      X[3:0]=4'b0011; Y[3:0]=4'b0011; Cin=0;  #5
      X[3:0]=4'b0111; Y[3:0]=4'b0111; Cin=0;  #5
      X[3:0]=4'b1111; Y[3:0]=4'b1111; Cin=0;  #5
      X[3:0]=4'b0000; Y[3:0]=4'b0000; Cin=1;  #5
      X[3:0]=4'b0001; Y[3:0]=4'b0001; Cin=1;  #5
      X[3:0]=4'b0011; Y[3:0]=4'b0011; Cin=1;  #5
      X[3:0]=4'b0111; Y[3:0]=4'b0111; Cin=1;  #5
      X[3:0]=4'b1111; Y[3:0]=4'b1111; Cin=1;  #5

      $dumpfile("ripple_carry_adder.vcd");
      $dumpvars;
    $finish;
    end
endmodule
```

Code for Design:

```
module full_adder(
    input X, Y, Cin,
    output S, Cout
);

    assign S = X ^ Y ^ Cin;
  assign Cout = ((X ^ Y) & Cin) | (X & Y);
endmodule

module ripple_carry_adder(
  input [3:0] X,
  input [3:0] Y,
  input Cin,
  output [3:0] S,
  output Cout
);

    wire c1, c2, c3;
  full_adder fa0(X[0], Y[0], Cin, S[0], c1);
  full_adder fa1(X[1], Y[1], c1, S[1], c2);
  full_adder fa2(X[2], Y[2], c2, S[2], c3);
  full_adder fa3(X[3], Y[3], c3, S[3], Cout);

endmodule
```

Output:

```
X=0000 Y=0000 Cin=0 S=0000 Cout=0
X=0001 Y=0001 Cin=0 S=0010 Cout=0
X=0011 Y=0011 Cin=0 S=0110 Cout=0
X=0111 Y=0111 Cin=0 S=1110 Cout=0
X=1111 Y=1111 Cin=0 S=1110 Cout=1
X=0000 Y=0000 Cin=1 S=0001 Cout=0
X=0001 Y=0001 Cin=1 S=0011 Cout=0
X=0011 Y=0011 Cin=1 S=0111 Cout=0
X=0111 Y=0111 Cin=1 S=1111 Cout=0
X=1111 Y=1111 Cin=1 S=1111 Cout=1
```

Q2. Design D-flip flop and reuse it to implement 4-bit Johnson Counter

Code for Test-bench:

```
module testbench;
  reg clock, reset;
  wire T0,T1,T2,T3;

  JOHNSON dut(clock,reste,T0,T1,T2,T3);

  initial
    begin
      $monitor("clock=%b reset=%b T0=%b T1=%b T2=%b
T3=%b",clock,reste,T0,T1,T2,T3);
    end

      always begin
        #5 clock = ~clock;
    end

    initial begin
        clock = 0;
        reset = 1;
        #10 reset = 0;
       initial
    begin
      $monitor("clock=%b reset=%b T0=%b T1=%b T2=%b
T3=%b",clock,reste,T0,T1,T2,T3);
    end
      $dumpfile("JOHNSON.vcd");
      $dumpvars;
    $finish;
    end
endmodule
```

Code for Design:

```
module JOHNSON(
input clock,reset,
output T0,T1,T2,T3
);

DFF A(T0,!(T3),clock,reset);
DFF B(T1,T0,clock,reset);
DFF C(T2,T1,clock,reset);
DFF D(T3,T2,clock,reset);



endmodule
module DFF ( output reg Q, input D, Clk, rst);
always @ ( posedge Clk, negedge rst)
if (!rst) Q <= 1'b0;
else Q <= D;
endmodule
```

Q3. Reuse 2:1 Mux code to implement 8:1 Mux.

Code for Test-bench:

```
module testbench;
  reg [7:0] a;
  reg [2:0] s;
  wire out;
  mux_8x1 dur(a,s,out);
  initial
    begin
      $monitor("a=%b s=%b out=%b", a,s,out);
      a=8'b11001100;
      s[2]=0; s[1]=0; s[0]=0; #10
      s[2]=0; s[1]=0; s[0]=1; #10
      s[2]=0; s[1]=1; s[0]=0; #10
      s[2]=0; s[1]=1; s[0]=1; #10
      s[2]=1; s[1]=0; s[0]=0; #10
      s[2]=1; s[1]=0; s[0]=1; #10
      s[2]=1; s[1]=1; s[0]=0; #10
      s[2]=1; s[1]=1; s[0]=1; #10

            $dumpfile("mux_8x1.vcd");
      $dumpvars;
    $finish;
    end
endmodule
```

Code for Design:

```
module mux_2x1 (
    input a0,a1,s,
    output out
);
    wire sn,k1,k2;

  not(sn,s);
  and(k1,a0,sn);
  and(k2,a1,s);
  or(out,k1,k2);

endmodule

module mux_8x1 (
  input [7:0] a,
    input [2:0] s,
    output out
);
  wire k1,k2,k3,k4,k5,k6;


  mux_2x1 mux1(a[0],a[1],s[0],k1);
  mux_2x1 mux2(a[2],a[3],s[0],k2);
  mux_2x1 mux3(a[4],a[5],s[0],k3);
  mux_2x1 mux4(a[6 ],a[7],s[0],k4);
  mux_2x1 mux5(k1,k2,s[1],k5);
  mux_2x1 mux6(k3,k4,s[1],k6);
  mux_2x1 mux7(k5,k6,s[2],out);

endmodule
```

Output:

```
a=11001100 s=000 out=0
a=11001100 s=001 out=0
a=11001100 s=010 out=1
a=11001100 s=011 out=1
a=11001100 s=100 out=0
a=11001100 s=101 out=0
a=11001100 s=110 out=1
a=11001100 s=111 out=1
```

Q4. Design a Full Subtractor with Gate Level Modeling Style. (use primitive gates)

Code for Test-bench:

```
module testbench;
  reg A,B,BorrowIn;
  wire Diff,BorrowOut;

  full_subtractor_gate_level du(A,B,BorrowIn,Diff,BorrowOut);

  initial
    begin
      $monitor("A=%b B=%b BorrowIn=%b Diff=%b BorrowOut=%b",
A,B,BorrowIn,Diff,BorrowOut);

       A=0; B=0; BorrowIn=0; #10;
       A=0; B=0; BorrowIn=1; #10;
       A=0; B=1; BorrowIn=0; #10;
       A=0; B=1; BorrowIn=1; #10;
       A=1; B=0; BorrowIn=0; #10;
       A=1; B=0; BorrowIn=1; #10;
       A=1; B=1; BorrowIn=0; #10;
       A=1; B=1; BorrowIn=1; #10;


            $dumpfile("full_subtractor_gate_level.vcd");
      $dumpvars;
    $finish;
    end
endmodule
```

Code for Design:

```
module full_subtractor_gate_level (
    input A,
    input B,
    input BorrowIn,
    output Diff,
    output BorrowOut
);

  wire d1,b1,b2,b3,xn;
  xor(d1,A,B);
  xor(Diff,d1,BorrowIn);
  not(xn,A);
  and(b1,xn,B);
  and(b2,xn,BorrowIn);
  and(b3,B,BorrowIn);
  or(BorrowOut,b1,b2,b3);

endmodule
```

Output:

```
A=0 B=0 BorrowIn=0 Diff=0 BorrowOut=0
A=0 B=0 BorrowIn=1 Diff=1 BorrowOut=1
A=0 B=1 BorrowIn=0 Diff=1 BorrowOut=1
A=0 B=1 BorrowIn=1 Diff=0 BorrowOut=1
A=1 B=0 BorrowIn=0 Diff=1 BorrowOut=0
A=1 B=0 BorrowIn=1 Diff=0 BorrowOut=0
A=1 B=1 BorrowIn=0 Diff=0 BorrowOut=0
A=1 B=1 BorrowIn=1 Diff=1 BorrowOut=1
```

Q5. Design a 2X4 decoder using gate level modelling

Code for Test-bench:

```
module testbench;
  reg I0,I1;
  wire b0,b1,b2,b3;

  decoder_2x4_gate_level dut(I0,I1,b0,b1,b2,b3);

  initial
    begin
      $monitor("I0=%b I1=%b b0=%b b1=%b b2=%b b3=%b",I0,I1,b0,b1,b2,b3);

       I0=0; I1=0; #10;
       I0=0; I1=1; #10;
       I0=1; I1=0; #10;
       I0=1; I1=1; #10;



            $dumpfile("decoder_2x4_gate_level.vcd");
      $dumpvars;
    $finish;
    end
endmodule
```

Code for Design:

```
module decoder_2x4_gate_level (
    input I0,
    input I1,
    output b0,b1,b2,b3
);

   wire a0n,a1n;
  not(a0n,I0);
  not(a1n,I1);
  and(b0,a0n,a1n);
  and(b1,a0n,I1);
  and(b2,I0,a1n);
  and(b3,I0,I1);

endmodule
```

Output:

```
I0=0 I1=0 b0=1 b1=0 b2=0 b3=0
I0=0 I1=1 b0=0 b1=1 b2=0 b3=0
I0=1 I1=0 b0=0 b1=0 b2=1 b3=0
I0=1 I1=1 b0=0 b1=0 b2=0 b3=1
```

Q6. Design a 4x1 mux using operators. (use data flow)

Code for Test-bench:

```
module testbench;
  reg [3:0] data;
  reg s0,s1;
  output out0,out1,out2,out3;

  mux_4x1_data_flow dut(data,s0,s1,out0,out1,out2,out3);

  initial
    begin
      $monitor("data=%b s0=%b s1=%b out0=%b out1=%b out2=%b out3=%b",
data,s0,s1,out0,out1,out2,out3);

     data=4'b1100;
      s0=0; s1=0; #5;
      s0=0; s1=1; #5;
      s0=1; s1=0; #5;
      s0=1; s1=1; #5;

      $dumpfile("mux_4x1_data_flow.vcd");
      $dumpvars;
    $finish;
    end
endmodule
```

Code for Design:

```
module mux_4x1_data_flow (
  input [3:0] data,
    input s0,s1,
  output out0,out1,out2,out3
);

   assign out0 = ~s1 & ~s0;
   assign out1 = s1 & ~s0;
   assign out2 = ~s1 & s0;
   assign out3 = s1 & s0;


endmodule
```

Output:

```
data=1100 s0=0 s1=0 out0=1 out1=0 out2=0 out3=0
data=1100 s0=0 s1=1 out0=0 out1=1 out2=0 out3=0
data=1100 s0=1 s1=0 out0=0 out1=0 out2=1 out3=0
data=1100 s0=1 s1=1 out0=0 out1=0 out2=0 out3=1
```

Q7. Design a Full adder using half adder

Code for Test-bench:

```
module testbench;
  reg A,B,Cin;
  wire Sum,Cout;

  full_adder dut(A,B,Cin,Sum,Cout);

  initial
    begin
      $monitor("A=%b B=%b Cin=%b Sum=%b Cout=%b",A,B,Cin,Sum,Cout);

      A=0; B=0; Cin=0; #5;
      A=0; B=0; Cin=1; #5;
      A=0; B=1; Cin=0; #5;
      A=0; B=1; Cin=1; #5;
      A=1; B=0; Cin=0; #5;
      A=1; B=0; Cin=1; #5;
      A=1; B=1; Cin=0; #5;
      A=1; B=1; Cin=1; #5;

      $dumpfile("full_adder.vcd");
      $dumpvars;
    $finish;
    end
endmodule
```

Code for Design:

```
module half_adder (
     input A,
     input B,
     output Sum,
     output Carry
);

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

endmodule

module full_adder (
     input A,
     input B,
     input Cin,
     output Sum,
     output Cout
);
  wire s1,c1,c2;
  half_adder ha1 (A,B,s1,c1);
  half_adder ha2 (Cin,s1,Sum,c2);

  or(Cout,c1,c2);


endmodule
```

Output

```
A=0 B=0 Cin=0 Sum=0 Cout=0
A=0 B=0 Cin=1 Sum=1 Cout=0
A=0 B=1 Cin=0 Sum=1 Cout=0
A=0 B=1 Cin=1 Sum=0 Cout=1
A=1 B=0 Cin=0 Sum=1 Cout=0
A=1 B=0 Cin=1 Sum=0 Cout=1
A=1 B=1 Cin=0 Sum=0 Cout=1
A=1 B=1 Cin=1 Sum=1 Cout=1
```