BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

Lab Sheet - 2 Topic: Combinational Circuit Modeling

Learning Objectives:

i) **Combinational digital circuit modeling**

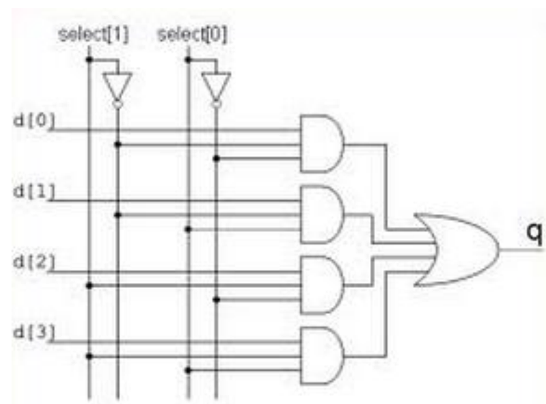
ii) **Constructing a larger module using a smaller module**

Introduction:

In the first lab, we have learned how to simulate digital circuits using different types of modeling (i.e. Gate level, Data flow, Behavioral) in VeriLog. In this lab we will do more practice on combinational circuit simulation. Also, we will learn the hierarchical modeling (i.e. implement a larger module using a smaller module).

Problem Description:

Implement a gate level modeling of a 4x1 multiplexer. Further using the 4x1 MUX module, implement a 16x1 multiplexer.



Sample Multiplexer Circuit

Solution:

```
module mux4to1_gate(out,in,sel);
input [0:3] in;
input [0:1] sel;
output out;
wire a,b,c,d,n1,n2,a1,a2,a3,a4;

not n(n1,sel[1]);
not nn(n2,sel[0]);

and (a1,in[0],n1,n2);
and (a2,in[1],n2,sel[1]);
and (a3,in[2],sel[0],n1);
and (a4,in[3],sel[0],sel[1]);

or or1(out,a1,a2,a3,a4);
endmodule

module mux16to1(out,in,sel);
input [0:15] in;
input [0:3] sel;
output out;
wire [0:3] ma;

mux4to1_gate mux1(ma[0],in[0:3],sel[2:3]);
mux4to1_gate mux2(ma[1],in[4:7],sel[2:3]);
mux4to1_gate mux3(ma[2],in[8:11],sel[2:3]);
mux4to1_gate mux4(ma[3],in[12:15],sel[2:3]);
mux4to1_gate mux5(out,ma,sel[0:1]);

endmodule
```

```
module testmux_16;

    reg [0:15] in;
    reg [0:3] sel;
    wire out;

    mux16to1 mux(out,in,sel);

initial
begin
$monitor("in=%b | sel=%b | out=%b",in,sel,out);
end

initial
begin

      in=16'b1000000000000000; sel=4'b0000;
   #3 in=16'b0100000000000000; sel=4'b0001;
   #3 in=16'b0010000000000000; sel=4'b0010;
   #3 in=16'b0001000000000000; sel=4'b0011;
   #3 in=16'b0000100000000000; sel=4'b0100;
   #3 in=16'b0000010000000000; sel=4'b0101;
   #3 in=16'b0000001000000000; sel=4'b0110;
   #3 in=16'b0000000100000000; sel=4'b0111;
   #3 in=16'b0000000010000000; sel=4'b1000;
   #3 in=16'b0000000001000000; sel=4'b1001;
   #3 in=16'b0000000000100000; sel=4'b1010;
   #3 in=16'b0000000000010000; sel=4'b1011;
   #3 in=16'b0000000000001000; sel=4'b1100;
   #3 in=16'b0000000000000100; sel=4'b1101;
   #3 in=16'b0000000000000010; sel=4'b1110;
   #3 in=16'b0000000000000001; sel=4'b1111;
end
endmodule
```

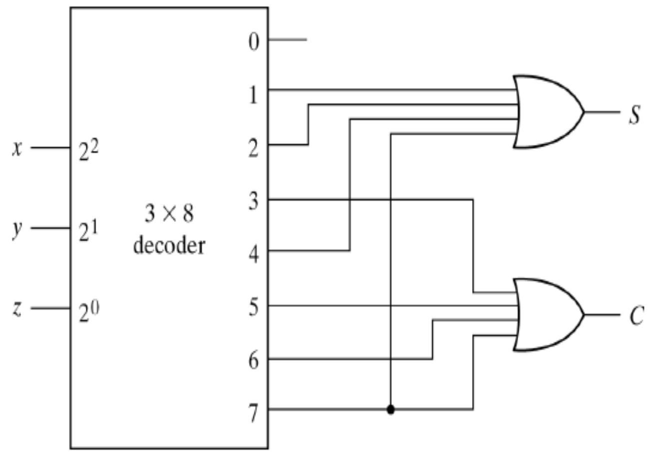
Exercise-1

Consider the circuit diagram for full adder using a 3x8 decoder given below.
It takes 3-bit input number and produce Sum and Carry bit as an output

Equation

$S(x, y, z) = \Sigma(1,2,4,7)$

$C(x, y, z) = \Sigma(3,5,6,7)$



Write Verilog code to implement a full adder as given in the above circuit

A. Write 3-to-8 decoder using Gate level model.

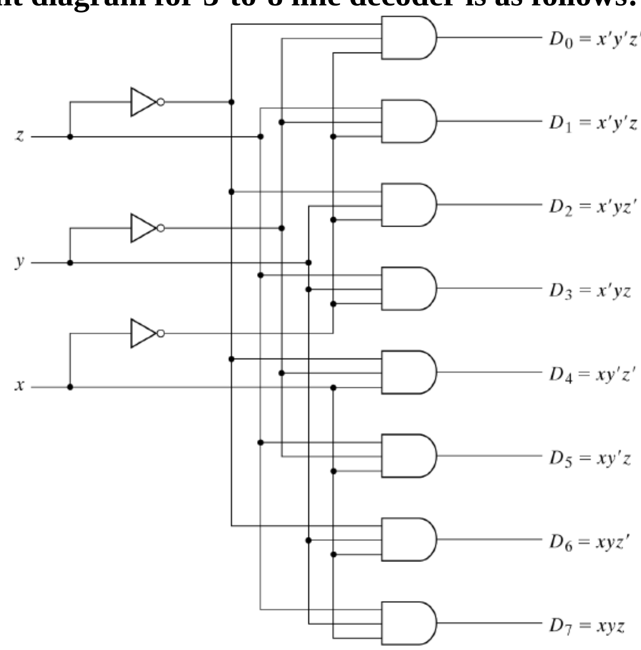
Name the module as DECODER( )

B. Write full adder using Data flow model

Name the module as FADDER( )

Write appropriate test bench module to check the correctness of your design. Specify all possible combinations in the **test bench**.

Circuit diagram for 3-to-8 line decoder is as follows:



Solution:

```
module DECODER(d0,d1,d2,d3,d4,d5,d6,d7,x,y,z);
```

```
input x,y,z;
output d0,d1,d2,d3,d4,d5,d6,d7;
wire x0,y0,z0;
not n1(x0,x);
not n2(y0,y);
not n3(z0,z);
and a0(d0,x0,y0,z0);
and a1(d1,x0,y0,z);
and a2(d2,x0,y,z0);
and a3(d3,x0,y,z);
and a4(d4,x,y0,z0);
and a5(d5,x,y0,z);
and a6(d6,x,y,z0);
and a7(d7,x,y,z);

endmodule

module FADDER(s,c,x,y,z);
input x,y,z;
wire d0,d1,d2,d3,d4,d5,d6,d7;
output s,c;
DECODER dec(d0,d1,d2,d3,d4,d5,d6,d7,x,y,z);
assign s = d1 | d2 | d4 | d7,
       c = d3 | d5 | d6 | d7;

endmodule

module testbench;
    reg x,y,z;
    wire s,c;
    FADDER fl(s,c,x,y,z);
    initial
       $monitor(,$time,"x=%b,y=%b,z=%b,s=%b,c=%b",x,y,z,s,c);
    initial
       begin
       #0 x=1'b0;y=1'b0;z=1'b0;
       #4 x=1'b1;y=1'b0;z=1'b0;
       #4 x=1'b0;y=1'b1;z=1'b0;
       #4 x=1'b1;y=1'b1;z=1'b0;
       #4 x=1'b0;y=1'b0;z=1'b1;
       #4 x=1'b1;y=1'b0;z=1'b1;
       #4 x=1'b0;y=1'b1;z=1'b1;
       #4 x=1'b1;y=1'b1;z=1'b1;
      end
endmodule
```

Task to do:

~~i)~~ **Implement a 8-bit adder circuit using the above 1-bit adder module.**

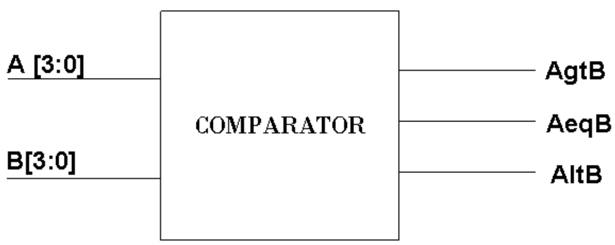
~~ii)~~ **Implement a 32-bit adder using 8-bit adder module.**

Exercise-2

Write a VeriLog code (behavioral modeling) for the simulation of a comparator which compares two signed 4-bit numbers A, and B and generates three output lines (as shown in Fig.1) one each for A equals to B (A==B), A greater than B(A>B) and A less than B (A<B). Only one of the three output lines must be active for a given input values of A and B. Assume that A and B are represented as 2's complement numbers.
Also write one test bench module to test the behavioral model of the comparator.

Test Cases:

0 A=0000, B=0000 AgtB=0, AeqB=1, AltB=0
1 A=1000, B=1011 AgtB=0, AeqB=0, AltB=1
2 A=0010, B=0111 AgtB=0, AeqB=0, AltB=1
3 A=0101, B=1111 AgtB=1, AeqB=0, AltB=0




```
module signa(neg,A);
//this module is used to get the sign of an input 4-digit number
input  [3:0] A;
output neg;
reg neg;

always @ (A)
    if (A[3] == 1)
     begin
     neg =1;
     end
     else
     neg =0;

endmodule
//*****************************************************************
module compar(A,B,signA,signB,CMP1, CMP2,CMP3);
//This module implement the compare code for input of two 3-digit
numbers using signa().
input  [3:0] A;
input  [3:0] B;
output signA,signB,CMP1,CMP2,CMP3;
reg CMP1,CMP2,CMP3;
signa forA(signA,A);
```

```
signa forB(signB,B);

always @ (A or B or signA or signB)// performs check for four different cases

      if(signA==1 && signB==0)
      begin
      CMP1 = 0;
    CMP2 = 0;
    CMP3 = 1;
      end

      else if(signA==0 && signB==1)
      begin
      CMP1 = 1;
    CMP2 = 0;
    CMP3 = 0;
      end

      else if (A > B  )
            begin
            CMP1 = 1;
         CMP2 = 0;
         CMP3 = 0;
            end

            else if (A == B)
            begin
            CMP1 = 0;
         CMP2 = 1;
         CMP3 = 0;
            end

            else
            begin
            CMP1 = 0;
         CMP2 = 0;
         CMP3 = 1;
            end
  endmodule
//*****************************************************************
module testbench;
//This module tests the functionality of compare() module
      reg Input,Clk;
      wire Out;
      reg [3:0] A;
      wire a,b,c,OutA,OutB,signA,signB,CMP1,CMP2,CMP3;
      reg [3:0] B;

     initial
      begin
            A=4'b0000;//input1
```

```
        B=4'b0000;//input2
    end

initial
    begin
        #1 A=-8;B=-5;
        #1 A=2; B=7;
        #1 A=5; B=-1;
end
    compar c1(A,B,signA,signB,CMP1,CMP2,CMP3);  //make        an
instance of compar()
initial
    begin
$monitor($time,"A=%b, B=%b AgrB=%b, AeqB=%b,
AltB=%b",A,B,CMP1,CMP2,CMP3);
end

initial
    begin
    #5 $finish;
end
endmodule
```

Exercise-3

Consider the given circuit diagram which performs add and subtract operations. The Input M controls the operation (M=1 for subtract and M=0 for the add operation). Output V=1, if overflow occurs otherwise V=0.

Tasks to be done:

~~i)~~ **Implement a FULLADDER() module using behavioral modeling.**

~~ii)~~ **Use FULLADDER() to implement ADDSUB() module which takes two 4-bit numbers and one control input M as input argument and produces an 4-bit sum/sub with an overflow bit V.**