VLSI LAB ASSIGNMENT-1

Harshavardhan Alimi

18EC10021

Aim :-

Part-1 :- To Design a 8:1 MUX (3 select inputs) using 2:1 MUXs (1 select input each). Write using both Behavioral and Structural styles.Demonstrate through the test bench waveform, that each combination of select inputs passes the correct input line to the output.

Part-2 :- To Implement a 4-bit ripple carry adder using 4 1-bit full adders. Demonstrate the output of the full adder using a test bench waveform.

Schematic diagrams and labels :-

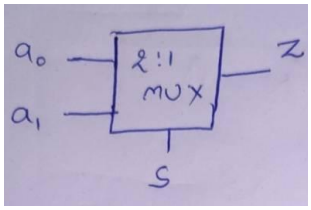


Fig.1:- 2:1 MULTIPLEXER MODEL

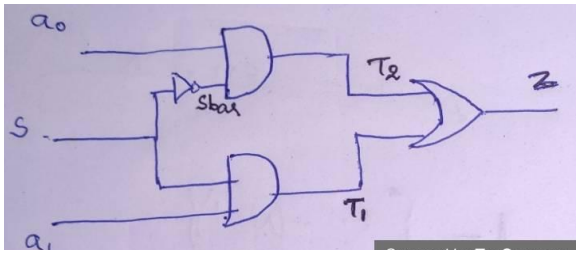


Fig.2 :- Block diagram for Structural implementation of 2:1 MUX

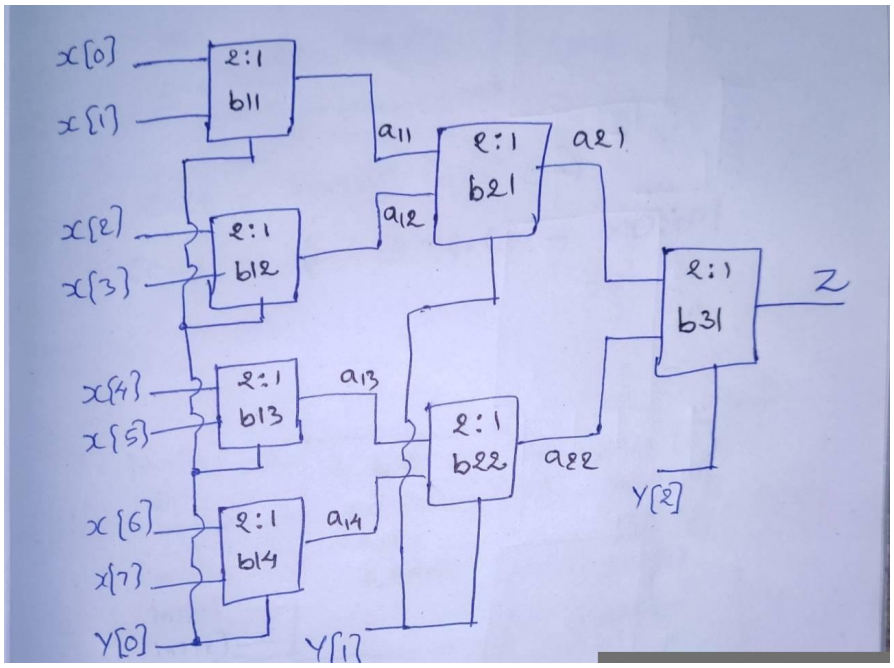


Fig.3 :- Block diagram for Structural implementation of 8:1 MUX using only 2:1 MUX model

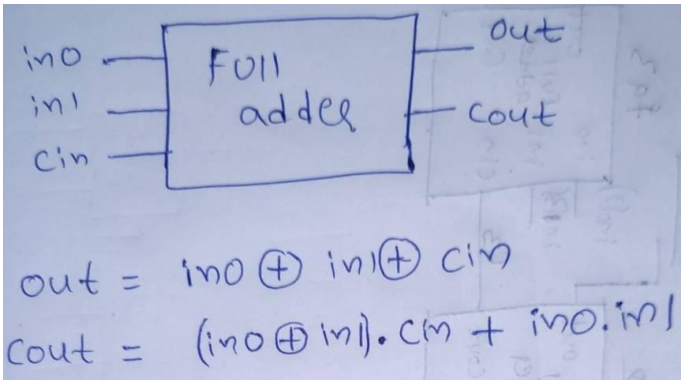


Fig.4 :- Behavioural implementation of Full adder

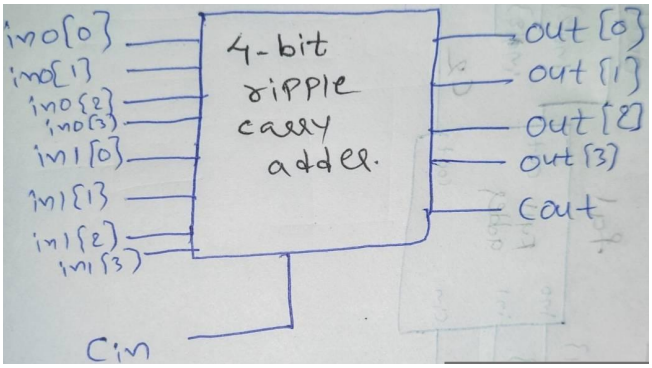


Fig.5 :- 4-bit ripple carry adder model

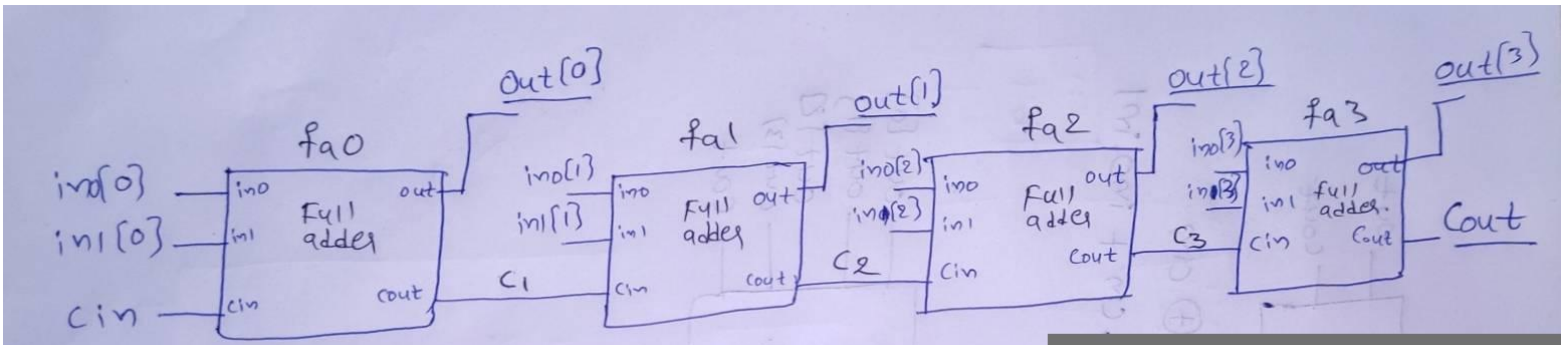


Fig.6 :-Block diagram for Structural implementation of 4-bit ripple carry adder using only full adder model

Description :-

Part-1

- Behavioural implementation of 2:1 MUX module can be done by assigning the value `((~s)&a0) | (s&a1)` to the output of the MUX,its structural module can be implemented as shown in the Fig.2
- These 2:1 multiplexers are used as building blocks for the 8:1 multiplexers Here,it uses **7** 2:1 multiplexers to implement the 8:1 multiplexer,This implementation of multiplexing can be done in 3 stages as shown in the Fig.3.
- 8:1 MUX module takes one 8-bit input along with a 3-bit select input.
- As the select input varies from 000 to 111, each bit of the input , from least significant to the most significant bit, becomes the output of the 8:1 MUX. (Example: consider the input IN7 IN6 IN5 IN4 IN3 IN2 IN1 IN0 ,when SEL =000, IN0 is the output, when SEL =001, IN1 is the output and so on.)
- In my code,the input variable is x,select line is y and output is wire z.

Part-2

- Full adder can be implemented by using the above logic stated in the Fig.4
- Implementation of 4-bit ripple carry adder requires **4** Full adders, the carry out (cout) of every Full adder will be taken as carry in(Cin) of the next stage,due to which we call this implementation as ripple carry adder as shown in the Fig.6.
- In my code,the input is 4-bit in04-bit ,in1 with 1-bit cin as carry input and output as 4-bit out and 1-bit cout(carry out).

Verilog codes :-

Part-1

Behavioural module :-

```
module design_2_1_MUX(
       input a0,
       input a1,
       input s,
       output z
       );
   assign z = ((~s)&a0) | (s&a1) ;
endmodule
module design_8_1_MUX(
       input [7:0] x,
       input [2:0] y,
       output z
       );
   wire a11,a12,a13,a14,a21,a22;
   design_2_1_MUX b11(x[0],x[1],y[0],a11);
   design_2_1_MUX b12(x[2],x[3],y[0],a12);
   design_2_1_MUX b13(x[4],x[5],y[0],a13);
   design_2_1_MUX b14(x[6],x[7],y[0],a14);
   design_2_1_MUX b21(a11,a12,y[1],a21);
   design_2_1_MUX b22(a13,a14,y[1],a22);
   design_2_1_MUX b31(a21,a22,y[2],z);
endmodule
```

```
module behavioral_8_1_MUX;
   // Inputs
   reg [7:0] x;
   reg [2:0] y;
   // Outputs
   wire z;
   // Instantiate the Unit Under Test (UUT)
   design_8_1_MUX uut (
       .x(x),
       .y(y),
       .z(z)
   );
   initial begin
       $dumpfile("test.vcd");
       $dumpvars(0,behavioral_8_1_MUX);
   end
   initial begin
   #0  x = 8'b10101010;y = 3'b000;
   #10 x = 8'b10101010;y = 3'b001;
   #10 x = 8'b10101010;y = 3'b010;
   #10 x = 8'b10101010;y = 3'b011;
   #10 x = 8'b10101010;y = 3'b100;
   #10 x = 8'b10101010;y = 3'b101;
   #10 x = 8'b10101010;y = 3'b110;
   #10 x = 8'b10101010;y = 3'b111;
   #10;
   end
```

```
   initial begin
       $monitor("t=%3d x=%b,y=%b,z=%b \n",$time,x,y,z);
   end
endmodule
```

Structural module :-.

```
module design_2_1_MUX(
       input a0,
       input a1,
       input s,
       output z
       );
   wire T1,T2,sbar;
   not(sbar,s);
   and (T1,a1,s) , (T2,a0,sbar);
   or (z,T1,T2);
endmodule
module design_8_1_MUX(
       input [7:0] x,
       input [2:0] y,
       output z
       );
   wire a11,a12,a13,a14,a21,a22;
   design_2_1_MUX b11(x[0],x[1],y[0],a11);
   design_2_1_MUX b12(x[2],x[3],y[0],a12);
   design_2_1_MUX b13(x[4],x[5],y[0],a13);
   design_2_1_MUX b14(x[6],x[7],y[0],a14);
```

```
   design_2_1_MUX b21(a11,a12,y[1],a21);
   design_2_1_MUX b22(a13,a14,y[1],a22);
   design_2_1_MUX b31(a21,a22,y[2],z);
endmodule
module structural_8_1_MUX;
   // Inputs
   reg [7:0] x;
   reg [2:0] y;
   // Outputs
   wire z;
   // Instantiate the Unit Under Test (UUT)
   design_8_1_MUX uut (
       .x(x),
       .y(y),
       .z(z)
   );
   initial begin
       $dumpfile("test.vcd");
       $dumpvars(0,structural_8_1_MUX);
   end
   initial begin
   // Initialize Inputs
   #0  x = 8'b10101010;y = 3'b000;
   #10 y = 3'b001;
   #10 y = 3'b010;
   #10 y = 3'b011;
   #10 y = 3'b100;
```

```
    #10 y = 3'b101;
    #10 y = 3'b110;
    #10 y = 3'b111;
    #10;
    end
    initial begin
        $monitor("t=%3d x=%b,y=%b,z=%b \n",$time,x,y,z);
    end
endmodule
```

Part-2

```
module full_adder(in0, in1, cin, out, cout);
    input in0, in1, cin;
    output out, cout;

    assign out = in0 ^ in1 ^ cin;
    assign cout = ((in0 ^ in1) & cin) | (in0 & in1);
endmodule
module ripple_carry_adder(in0, in1,cin, out, cout);
    input [3:0] in0;
    input [3:0] in1;
    input cin;
    output [3:0] out;
    output cout;
    wire c1, c2, c3;
    full_adder fa0(in0[0], in1[0], cin, out[0], c1);
    full_adder fa1(in0[1], in1[1], c1, out[1], c2);
```

```
   full_adder fa2(in0[2], in1[2], c2, out[2], c3);
   full_adder fa3(in0[3], in1[3], c3, out[3], cout);
endmodule
module ripple_carry_adder_4bit;
   // Inputs
   reg [3:0] in0;
   reg [3:0] in1;
   reg cin;
   // Outputs
   wire [3:0] out;
   wire cout;
   // Instantiate the Unit Under Test (UUT)
   ripple_carry_adder uut (.in0(in0), .in1(in1),.cin(cin), .out(out),
.cout(cout));
   initial begin
       $dumpfile("test.vcd");
       $dumpvars(0,ripple_carry_adder_4bit);
   end
   initial begin
   // Initialize Inputs
       #0  cin =1'b0 ; in0 = 4'b0110; in1 = 4'b0000;
       #10 in0 = 4'b0110; in1 = 4'b0001;
       #10 in0 = 4'b0110; in1 = 4'b0010;
       #10 in0 = 4'b0110; in1 = 4'b0011;
       #10 in0 = 4'b0110; in1 = 4'b0100;
       #10 in0 = 4'b0110; in1 = 4'b0101;
       #10 in0 = 4'b0110; in1 = 4'b0110;
       #10 in0 = 4'b0110; in1 = 4'b0111;
```

```
    #10 in0 = 4'b0110; in1 = 4'b1000;
    #10 in0 = 4'b0110; in1 = 4'b1001;
    #10 in0 = 4'b0110; in1 = 4'b1010;
    #10 in0 = 4'b0110; in1 = 4'b1011;
    #10 in0 = 4'b0110; in1 = 4'b1100;
    #10 in0 = 4'b0110; in1 = 4'b1101;
    #10 in0 = 4'b0110; in1 = 4'b1110;
    #10 in0 = 4'b0110; in1 = 4'b1111;
    #10; cin=1'b1; in0 = 4'b0110; in1 = 4'b0000;
    #10 in0 = 4'b0110; in1 = 4'b0001;
    #10 in0 = 4'b0110; in1 = 4'b0010;
    #10 in0 = 4'b0110; in1 = 4'b0011;
    #10 in0 = 4'b0110; in1 = 4'b0100;
    #10 in0 = 4'b0110; in1 = 4'b0101;
    #10 in0 = 4'b0110; in1 = 4'b0110;
    #10 in0 = 4'b0110; in1 = 4'b0111;
    #10 in0 = 4'b0110; in1 = 4'b1000;
    #10 in0 = 4'b0110; in1 = 4'b1001;
    #10 in0 = 4'b0110; in1 = 4'b1010;
    #10 in0 = 4'b0110; in1 = 4'b1011;
    #10 in0 = 4'b0110; in1 = 4'b1100;
    #10 in0 = 4'b0110; in1 = 4'b1101;
    #10 in0 = 4'b0110; in1 = 4'b1110;
    #10 in0 = 4'b0110; in1 = 4'b1111;
    #10;
end
```

```
   initial begin
       $monitor("t=%3d out=%b cout=%b\n",$time,out,cout);
   end
endmodule
```

Runtime log :-

Part-1

Behavioural module :-

VCD info: dumpfile test.vcd opened for output.

t= 0 x=10101010,y=000,z=0

t= 10 x=10101010,y=001,z=1

t= 20 x=10101010,y=010,z=0

t= 30 x=10101010,y=011,z=1

t= 40 x=10101010,y=100,z=0

t= 50 x=10101010,y=101,z=1

t= 60 x=10101010,y=110,z=0

t= 70 x=10101010,y=111,z=1

Structural module :-.

VCD info: dumpfile test.vcd opened for output.

t= 0 x=10101010,y=000,z=0

t= 10 x=10101010,y=001,z=1

t= 20 x=10101010,y=010,z=0

t= 30 x=10101010,y=011,z=1

t= 40 x=10101010,y=100,z=0

t= 50 x=10101010,y=101,z=1

t= 60 x=10101010,y=110,z=0

t= 70 x=10101010,y=111,z=1

Part-2

VCD info: dumpfile test.vcd opened for output.

t= 0 out=0110 cout=0

t= 10 out=0111 cout=0

t= 20 out=1000 cout=0

t= 30 out=1001 cout=0

t= 40 out=1010 cout=0

t= 50 out=1011 cout=0

t= 60 out=1100 cout=0

t= 70 out=1101 cout=0

t= 80 out=1110 cout=0

t= 90 out=1111 cout=0

t=100 out=0000 cout=1

t=110 out=0001 cout=1

t=120 out=0010 cout=1

t=130 out=0011 cout=1

t=140 out=0100 cout=1

t=150 out=0101 cout=1

t=160 out=0111 cout=0

t=170 out=1000 cout=0

t=180 out=1001 cout=0

t=190 out=1010 cout=0

t=200 out=1011 cout=0

t=210 out=1100 cout=0

t=220 out=1101 cout=0

t=230 out=1110 cout=0

t=240 out=1111 cout=0

t=250 out=0000 cout=1

t=260 out=0001 cout=1

t=270 out=0010 cout=1

t=280 out=0011 cout=1

t=290 out=0100 cout=1

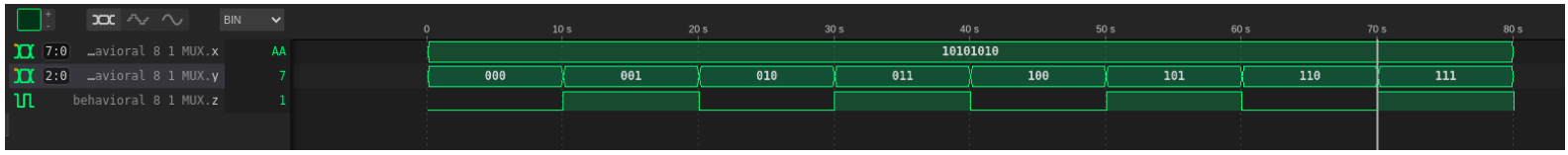
t=300 out=0101 cout=1

t=310 out=0110 cout=1

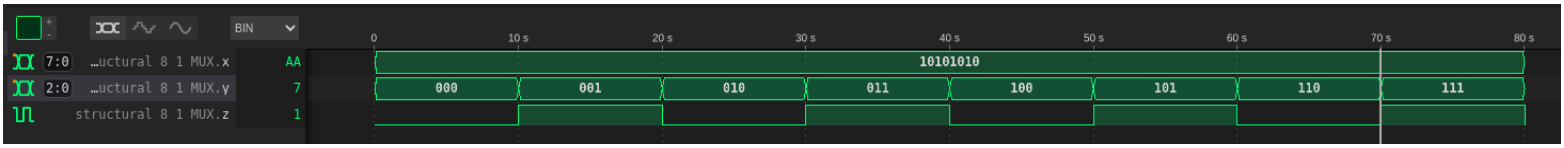
Wave forms :-

Part-1

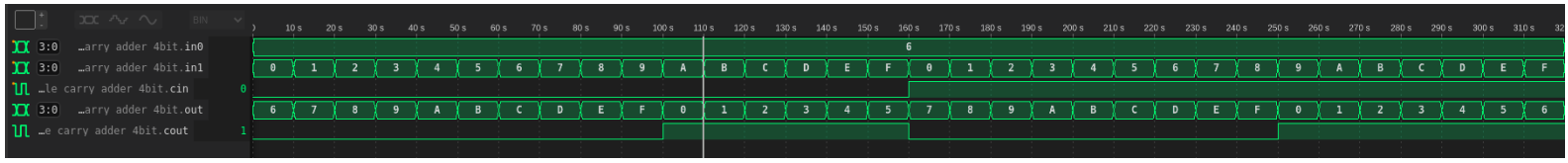
Behavioural module :-



Structural module :-.



Part-2



Conclusion :-

❖ Part-1

- ❖ Behavioural implementation of 2:1 MUX is made using logic operations of the verilog and assigning the value of that operations to the output wire,whereas structural implementation uses in-built logic gates such as NOT,AND,OR gates for implementation.

- The output wave forms were generated using GTKWAVE and it has been observed for all 8 values of the select line keeping the 8-bit input x constant to show the working of the 8:1 MUX.

Part-2

- The full adder module was implemented in behavioral style as it uses the logic operations to assign the values for the output.
- The full adder module designed is used to design the 4-bit ripple carry adder.
- The output has been generated using the GTKWAVE. One of the inputs(in0) is set at 6 throughout the testing and the other input(in1) is incremented from from 6 upto F(15) and also seen the effect of changing the cin (0/1) on the out and cout.

Google Drive Links:-(for verilog codes)

Behavioural_8_1_MUX

Structural_8_1_MUX

Ripple_carry_adder_4bit