Digital Logic Design Experiment 3

Name: SPARSH ARYA

Registration Number: 17BEC0656

Full adder:

```
module FA(a,b,c,s,cout);
innput a,b,c;
output s,cout;
assign s=a^b^c;
assign cout=(a&b)|(c&(a^b));
endmodule
```

Full adder testbench:

```
module FA_tb;
reg a,b,c;
wire s,cout;
FA FL (a,b,c,s,cout);
initial begin
a=1'b0;b=1'b0;c=1'b0;
#50 a=1'b0;b=1'b0;c=1'b1;
#50 a=1'b0;b=1'b1;c=1'b0;
#50 a=1b'0;b=1'b1;c=1'b1;
#50 a=1'b1;b=1'b0;c=1'b0;
#50 a=1'b1;b=1'b0;c=1'b1;
#50 a=1'b1;b=1'b1;c=1'b0;
#50 a=1'b1;b=1'b1;c=1'b1;
end
initial
$monitor($time,"a=%b,b=%b,c=%b,s=%b,cout=%b", a,b,c,s,cout);
initial
#1000 $stop;
Endmodule
```

Ripple carry order:

```
module RCA(a,b,c,s,cout);
input [3:0] a,b;
input c;
output [3:0] s;
output cout;
FA FL(a[0],b[0],c,s,c1);
FA F2(a[1],b[1],c1,s[1],c2);
FA F3(a[2],b[2],c2,s[2],c3);
FA F4(a[3],b[3],c3,s[3],cout);
endmodule
```

Ripple carry order testbench:

```
module rca_tb;
 reg [3:0] A;
 reg [3:0] B;
 reg Cin;
 wire [3:0] Sum;
 wire Cout;
 ripple_adder_4bit uut (
  .Sum(Sum),
  .Cout(Cout),
  .A(A),
  .B(B),
  .Cin(Cin)
 );
 initial begin
  A = 0;
  B = 0;
  Cin = 0;
  #100;
  A=4'b0001;B=4'b0000;Cin=1'b0;
  #10 A=4'b1010;B=4'b0011;Cin=1'b0;
  #10 A=4'b1101;B=4'b1010;Cin=1'b1;
 end
 initial begin
  $monitor("time=",$time,, "A=%b B=%b Cin=%b : Sum=%b Cout=%b",A,B,Cin,Sum,Cout);
 end

endmodule
```

Multiplexer:

```
module mux1( select, d, q );
input[1:0] select;
input[3:0] d;
output    q;
wire     q;
wire[1:0] select;
wire[3:0] d;
assign q = d[select];
endmodule
```

Multiplexer testbench:

```
module mux_tb;
reg[3:0] d;
reg[1:0] select;
wire    q;
```

```
integer i;
mux1 my_mux( select, d, q );
initial
begin
  #1 $monitor("d = %b", d, " | select = ", select, " | q = ", q );

  for( i = 0; i <= 15; i = i + 1)
  begin
    d = i;
    select = 0;  #1;
    select = 1;  #1;
    select = 2;  #1;
    select = 3;  #1;
    $display("-----------------------------------------");
   ends

end
endmodule
```

Priority encoder:

```
module pr_en ( input [7:0] a,
          input [7:0] b,
          input [7:0] c,
          input [7:0] d,
          input [1:0] sel,
          output reg [7:0] out);

  always @ (a or b or c or d or sel) begin
    if (sel == 2'b00)
      out <= a;
    else if (sel == 2'b01)
      out <= b;
    else if (sel == 2'b10)
      out <= c;
    else
      out <= d;
  end
endmodule
```

Priority encoder test bench:

```
module tb_4to1_mux;
  reg [7:0] a;
  reg [7:0] b;
  reg [7:0] c;
  reg [7:0] d;
  wire [7:0] out;
  reg [1:0] sel;
  integer i;

  pr_en   pr_en0 (  .a (a),
                .b (b),
                .c (c),
                .d (d),
                .sel (sel),
                .out (out));

  initial begin
    sel <= 0;
    a <= $random;
    b <= $random;
    c <= $random;
    d <= $random;

    for (i = 1; i < 4; i=i+1) begin
      #5 sel <= i;
    end

    #5 $finish;
  end
endmodule
```

Comparator:

```
module comparator(
   Data_in_A,
   Data_in_B,
   less,
   equal,
    greater
   );
```

```
  input [3:0] Data_in_A;
  input [3:0] Data_in_B;
  output less;
   output equal;
   output greater;
   reg less;
   reg equal;
   reg greater;
  always @(Data_in_A or Data_in_B)
   begin
    if(Data_in_A > Data_in_B)   begin
      less = 0;
      equal = 0;
      greater = 1;   end
    else if(Data_in_A == Data_in_B) begin
      less = 0;
      equal = 1;
      greater = 0;   end
    else   begin
      less = 1;
      equal = 0;
      greater =0;
    end
  end
endmodule
```

Comparator testbench:

```
module tb_tm;

  reg [3:0] Data_in_A;
  reg [3:0] Data_in_B;

  wire less;
  wire equal;
  wire greater;

  comparator uut (
    .Data_in_A(Data_in_A),
    .Data_in_B(Data_in_B),
    .less(less),
    .equal(equal),
    .greater(greater)
  );
  initial begin
    Data_in_A = 10;
    Data_in_B = 12;
    #100;
    Data_in_A = 15;
    Data_in_B = 11;
    #100;
    Data_in_A = 10;
    Data_in_B = 10;
    #100;
  end

endmodule
```

END.