B210517CS - Binshadh Basheer

BEHAVIORAL

1) 4-WAY DEMULTIPLEXER

```
module conv(output reg [3:0]y,input [1:0]a,input d);
always@(a,d)begin
case(a)
    2'b00:{y}={1'b0,1'b0,1'b0,1'b0};
            2'b01:{y}={d,1'b0,1'b0,1'b0};
            2'b10:{y}={1'b0,d,1'b0,1'b0};
            2'b11:{y}={1'b0,1'b0,d,1'b0};
            default:{y}={1'b0,1'b0,1'b0,d};

            endcase
            end
            endmodule
```

TEST BENCH

```
module demultiplexertb;
reg [3:0]a;
reg d;
wire [3:0]y;
demultiplexer ins(y,a,d);
initial begin
d=1'b1;
integer i;
a=4'b0000;
for(i=0;i<16;i=i+1)begin

 a=a+i;
#10;
 end
 end
 endmodule
```

2) 8-WAY DEMULTIPLEXER

```
module conv(output reg [7:0]y,input [2:0]a,input d);
always@(a,d)begin
case(a)
    2'b000:{y}={1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,d};
           2'b001:{y}={d,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0 };
           2'b010:{y}={1'b0,d,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0 };
           2'b011:{y}={1'b0,1'b0,d,1'b0,1'b0,1'b0,1'b0,1'b0 };
           2'b100:{y}={1'b0,1'b0,1'b0,d,1'b0,1'b0,1'b0,1'b0 };
        2'b101:{y}={1'b0,1'b0,1'b0,1'b0,d,1'b0,1'b0,1'b0 };
```

```
2'b110:{y}={1'b0,1'b0,1'b0,1'b0,1'b0,d,1'b0,1'b0 };
2'b111:{y}={1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,d,1'b0 };

        endcase
        end
        endmodule
```

TEST BENCH

```
module test;
reg [2:0]a;
reg d;
wire [7:0]y;
conv ins(y,a,d);
initial begin
a=3’b000;
repeat(100)begin
a=a+{3’b001};
d=$random();
#10;
end
end
endmodule
```

3. 3:8 Decoder with Enable input (Active LOW)

```
module conv(output  reg d0,d1,d2,d3,d4,d5,d6,d7,input e,input [2:0]a);

always@(e,a)begin

         if(e==1'b0)begin
          case(a)
                3'b000:
{d0,d1,d2,d3,d4,d5,d6,d7}={1'b1,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0};
                3'b001:
{d0,d1,d2,d3,d4,d5,d6,d7}={1'b0,1'b1,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0};
                3'b010:
{d0,d1,d2,d3,d4,d5,d6,d7}={1'b0,1'b0,1'b1,1'b0,1'b0,1'b0,1'b0,1'b0};
                3'b011:
{d0,d1,d2,d3,d4,d5,d6,d7}={1'b0,1'b0,1'b0,1'b1,1'b0,1'b0,1'b0,1'b0};
                3'b100:
{d0,d1,d2,d3,d4,d5,d6,d7}={1'b0,1'b0,1'b0,1'b0,1'b1,1'b0,1'b0,1'b0};
                3'b101:
{d0,d1,d2,d3,d4,d5,d6,d7}={1'b0,1'b0,1'b0,1'b0,1'b0,1'b1,1'b0,1'b0};
                3'b110:
{d0,d1,d2,d3,d4,d5,d6,d7}={1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b1,1'b0};
                3'b111:
{d0,d1,d2,d3,d4,d5,d6,d7}={1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b1};
```

```
                    endcase
                    end
                   else begin
                     {d0,d1,d2,d3,d4,d5,d6,d7}={1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0};
          end


end
endmodule
```

TEST BENCH

```
test bench
module decodertb;
reg x,y,z,e;
wire d0,d1,d2,d3,d4,d5,d6,d7;
decoder ins(d0,d1,d2,d3,d4,d5,d6,d7,x,y,z,e);
initial begin
e=1'b0;
 x=1'b0;y=1'b0;z=1'b0;
repeat(8)begin
#10 {x,y,z}=$random();
end
end
endmodule
```

4. **4:16 Decoder with TWO 3:8 decoders with Enable input**

```
module decoder3to8(output d0,d1,d2,d3,d4,d5,d6,d7,input x,y,z,e);

assign
  d0=~e&~x&~y&~z,
  d1=~e&~x&~y&z,
  d2=~e&~x&y&~z,
  d3=~e&~x&y&z,
  d4=~e&x&~y&~z,
  d5=~e&x&~y&z,
  d6=~e&x&y&~z,
  d7=~e&x&y&z;
  endmodule

module conv(output reg[15:0]d,input x,y,z,e);
reg p=~e;
always@(x,y,z,e)begin

if(e==0)begin
```

```
decoder3to8 d1(d[0],d[1],d[2],d[3],d[4],d[5],d[6],d[7],x,y,z,p);
end
else begin
decoder3to8 d2(d[8],d[9],d[9],d[10],d[11],d[12],d[13],d[14],d[15],x,y,z,e);
end
end  endmodule
```

TEST BENCH

```
module testbesnch;
reg x,y,z,e;
wire [15:0]d;
conv ins(d,x,y,z,e);
initial begin
e=1'b0;
 x=1'b0;y=1'b0;z=1'b0;
repeat(8)begin
#10 x=$random();y=$random();z=$random();e=$random();
end
end
endmodule
```

5. 4-bit Binary Multiplier

```
module conv(output reg [3:0]y,input [3:0]a,b);
always@(a,b)begin
assign y=a*b;
end
endmodule
```

TEST BENCH

```
module test;
reg [3:0]a,b;
wire [3:0]y;
conv ins(y,a,b);
initial begin
a={4’b0000}
b={4’b0000}
integer i,j;
for (i=0;i<16,i=i+1)begin
b=b+{4’b0001};
for(j=0;j<16;j=j+1)begin
a=a+{4’b0001};
#10;
end
end
end
endmodule
```

6. Full adder using ONE Decoder and minimum number of OR gates

```
module conv(output reg s,cout,input x,y,z);
reg d0,d1,d2,d3,d4,d5,d6;
always@(x,y,z)begin

  d0=x&y&z;
  d1=~x&y&~z;
  d2=x&~y&~z;
  d3=~x&~y&z;
  d4=x&~y&z;
  d5=~x&y&z;
  d6=x&y&~z;
   s=d0|d2|d1|d3;
      cout=d0|d4|d5|d6;
end
endmodule
```

TEST BENCH

```
module testbench;
reg x,y,z,;
wire s,cout;
conv ins(s,x,y,z);
initial begin
{x,y,z}={3'b000};
integer i;
for(i=0;i<16;i=i+1)begin
{x,y,z}={x,y,z}+{3'b001};
#10;
end
end
endmodule
```

7. 8/16 - bit Binary - Gray (Gray- Binary) Code converters

```
module conv(output reg y1,y2,y3,y4,input [3:0]a);
always@(a[0],a[1],a[2],a[3])begin
case(a)
4'b0000:begin{y1,y2,y3,y4}={4'b0000};end//0
4'b0001:begin{y1,y2,y3,y4}={4'b0001};end//1
4'b0010:begin{y1,y2,y3,y4}={4'b0011};end//2
4'b0011:begin{y1,y2,y3,y4}={4'b0010};end//3
4'b0100:begin{y1,y2,y3,y4}={4'b0110};end//4
4'b0101:begin{y1,y2,y3,y4}={4'b0111};end//5
4'b0110:begin{y1,y2,y3,y4}={4'b0101};end//6
4'b0111:begin{y1,y2,y3,y4}={4'b0100};end//7
4'b1000:begin{y1,y2,y3,y4}={4'b1100};end//8
4'b1001:begin{y1,y2,y3,y4}={4'b1101};end//9
4'b1010:begin{y1,y2,y3,y4}={4'b1111};end//10
4'b1011:begin{y1,y2,y3,y4}={4'b1110};end//11
4'b1100:begin{y1,y2,y3,y4}={4'b1010};end//12
```

```
4'b1101:begin{y1,y2,y3,y4}={4'b1011};end//13
4'b1110:begin{y1,y2,y3,y4}={4'b0001};end//14
4'b1111:begin{y1,y2,y3,y4}={4'b1000};end//15

endcase
end
endmodule
```

TEST BENCH

```
module test;
reg a,b,c,d;
wire y1,y2,y3,y4;
conv ins(y1,y2,y3,y4,a,b,c,d);
initial begin
{a,b,c,d}={4’b0000};
repeat(16)begin
#10 {a,b,c,d}={a,b,c,d}+{4’b0001};

end
end
endmodule
```

8. Quadruple TWO-to-ONE Multiplexer

```
module conv(input enable,select,input [3:0]a,b ,output reg [3:0]y);
reg k1,k2;
always@(enable,select,a,b)begin
k1=~select&~enable;
k2=select&~enable;
   y[0]=(k1&a[0])|(k2&b[0]);
         y[1]=(k1&a[1])|(k2&b[1]);
         y[2]=(k1&a[2])|(k2&b[2]);
         y[3]=(k1&a[3])|(k2&b[3]);
end
 endmodule
```

TEST BENCH

```
module testbench;
reg enable ,select;
reg [3:0]a,b;
wire [3:0]y;
conv ins(enable,select,a,b,y);
initial begin
repeat(100)begin
enable=$random();
select=$rsndom();
a=$random%4;
b=$random%4;
#10;
end
end
endmodule
```

9. FOUR input Binary function F(w, x ,y, z)=∑m(1, 3, 5, 7, 8, 13, 14, 15) with 8:1 Multiplexer

```
module ques(input s3,s2,s1,s0,output reg out);
reg w0,w1,w2,w3,w4,w5,w6,w7;
always@(s3,s2,s1,s0,w0,w1,w2,w3,w4,w5,w6,w7)
begin
w0=s3;
w1=~s3;
w2=0;
w3=~s3;
w4=0;
w5=1;
w6=s3;
w7=1;

out = (w0 & ~s2 & ~s1 & ~s0) | (w1 & ~s2 & ~s1 & s0) | (w2 & ~s2 & s1 & ~s0) + (w3 & ~s2 &
s1 & s0) + (w4 & s2 & ~s1 & ~s0) + (w5 & s2 & ~s1 & s0) + (w6 & s2 & s1 & ~s0) + (w7 & s2 &
s1 & s0);

end
endmodule
```

TEST BENCH

```
module test;
reg s0,s1,s2,s3;
wire out;
ques ins(s3,s2,s1,s0,out)k
initial begin
{s3,s2,s1,s0}={4’b0000};
integer i;
for(i=0;i<16;i=i+1)begin
{s3,s2,s1,s0}={s3,s2,s1,s0}+{4’b0001};
end
end
endmodule
```

10. 4:1 MUX with Three state buffer and Enable 2:4 decoder

```
module conv(input a,b,c,d,e,s1,s0 ,output reg o);
always@(a,b,c,d,e,s1,s0)begin
if(e==1'b1)begin
o=1'b0;end
else if(s1==1'b0&&s0==1'b0) begin
o=a;
end
else if(s1==1'b0&&s0==1'b1) begin
o=b;
end
else if(s1==1'b1&&s0==1'b0) begin
o=c;
end
else  begin
o=d;
end
end
endmodule
```

TEST BENCH

```
module testbench;
reg   a,b,c,d,e,s1,s0;
wire o;
conv ins(a,b,c,d,e,s1,s0,o);
initial begin
repeat(100)begin
#100;
a=$random();
b=$random();
c=$random();
d=$random();
e=$random();
s1=$random();
s0=$random();
end
end
endmodule
```

11. Half subtractor

```
module conv(output reg diff,borrow,input a,b);
always@(a,b)begin
if(a==b)begin
  diff=1'b0;
  end
  else begin
  diff=1'b1;
  end
  end
always@(a,b)begin
if(a==1'b0&&b==1'b1)begin
 borrow=1'b1;
 end
 else begin borrow=1'b0;end

end
endmodule
```

<u>TEST BENCH</u>

```
module testbench;
reg a,b;
wire diff,borrow;
conv ins(diff,borrow,a,b);
initial begin
   a=0;b=0;
#5 a=0; b=1;
#5 a=1;b=0;
#5 a=1;a=1;
end
endmodule
```

<u>12. Full subtractor</u>

```
module conv(output  reg diff,borrow,input x,y,z);
always@(x,y,z)begin
if((x==1'b1&&y==1'b1&&z==1'b1)|(x==1'b1&&y==1'b0&&z==1'b0)|
(x==1'b0&&y==1'b1&&z==1'b0)|(x==1'b0&&y==1'b0&&z==1'b1))begin
 diff=1'b1; end
 else begin
 diff=1'b0; end
 end
always@(x,y,z)begin
 if((x==1'b1&&y==1'b1&&z==1'b1)|(x==1'b0&&y==1'b0&&z==1'b1)|
(x==1'b0&&y==1'b1&&z==1'b0)|(x==1'b0&&y==1'b1&&z==1'b1))begin
 borrow=1'b1;end
 else begin
 borrow =1'b0;
 end
 end

endmodule
```

TEST BENCH

```
module;
reg x,y,z;
wire diff,borrow;
conv ins(diff,borrow,x,y,z);
initial begin
{x,y,z}={3'b000};
integer i;
for(i=0;i<16;i=i+1)begin
{x,y,z}={x,y,z}+{3'b001};
#10;
end
end
endmodule
```

Q2: 4-bit arithmetic logic unit (ALU) performing the following operations:
i.
AND ii. OR iii. Add
iv. Subtract v. Less (If the first operand is less than the second
operand, output should be one, else the output should be zero.

```
module conv(output reg [3:0]y,output reg cout,input [2:0]s,input [3:0]a,b,input cin);
parameter
AND=3'b000,
OR=3'b001,
ADD=3'b010,
SUB=3'b011,
CMP=3'b100;

always@(a,b,s)begin
case(s)
AND:{cout,y}={1'b0,a&b};
OR:{cout,y}={1'b0,a|b};
ADD:{cout,y}={a+b+cin};
SUB:{cout,y}={a-b-cin};
default:{cout,y}={1'b0,((~a[3])&b[3])|(((~a[2])&b[2])&(~(a[3]^b[3])))|
(((~a[1])&b[1])&(~(a[3]^b[3]))&(~(a[2]^b[2])))|(((~a[0])&b[0])&(~(a[3]^b[3]))&(~(a[2]^b[2]))&(~(a[1]^b[1])))};
endcase
end
endmodule
```

TEST BENCH

```
module test;
reg [2:0]s;
reg [3:0]a,b;
reg cin;
```

```
wire [3:0]y;
wire cout;
conv ins(y,cout,s,a,b,cin);
initial begin
repeat(16)begin
#10;
s=$random%3;
a=$random%4;
b=$random%4;
cin=$random();
end
end
endmodule
```