BINSHADH BASHEER -B210517CS

—

DATA FLOW MODEL

1) 4-WAY DEMULTIPLEXER

```
module demultiplexer(output [3:0]y,input [1:0]a,input d);
wire w1,w2;
assign w1=~a[0],w2=~a[1];
assign y[0]=d&w1&w2,
      y[1]=d&w1&a[1],
                        y[2]=d&a[0]&w2,
                        y[3]=d&a[0]&a[1];
                        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

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

2) 8-WAY DEMULTIPLEXER

```
module demux(output [7:0]o,input d,input x,y,z);
assign
o[0]=d&~x&~y&~z,
o[1]=d&~x&~y&z,
o[2]=d&~x&y&~z,
o[3]=d&~x&y&z,
o[4]=d&x&~y&~z,
o[5]=d&x&~y&z,
o[6]=d&x&y&~z,
o[7]=d&x&y&z,
endmodule
```

TEST BENCH

```
module demuxtb;
reg d,x,y,z;
wire [7:0]o;
demux ins(o,x,y,z);
initial begin
{x,y,z}={3’b000};
integer i;
for(i=0;i<8;i=i+1)begin
#10 {x,y,z}={x,y,z}+{3’b001};
end
end
endmodule
```

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

```
module  decoder(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
```

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 [15:0]d,input x,y,z,e);
wire p=~e;
decoder3to8 d1(d[0],d[1],d[2],d[3],d[4],d[5],d[6],d[7],x,y,z,p);
decoder3to8 d2(d[8],d[9],d[9],d[10],d[11],d[12],d[13],d[14],d[15],x,y,z,e);
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 [3:0]y,input [3:0]a,b);
assign y=a*b;
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 s,cout,input x,y,z);
wire d0,d1,d3,d4,d5,d6;
assign
  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;
  assign s=d0|d2|d1|d3,
      cout=d0|d4|d5|d6;
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 y1,y2,y3,y4,input a,b,c,d);
assign y4=a,
       y3=a^b,
                    y2=b^c,
                    y1=c^d;
                    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
{a,b,c,d}={a,b,c,d}+{4’b0001};
#10;
end
end
endmodule
```

8. Quadruple TWO-to-ONE Multiplexer

```
module conv(input enable,select,input [3:0]a,b ,output [3:0]y);
wire k1,k2;
assign k1=~select      &~enable,k2=select&~enable;
assign
   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]);
          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 out);
wire w0,w1,w2,w3,w4,w5,w6,w7;
assign w0=s3,
w1=~s3,
w2=0,
w3=~s3,
w4=0,
w5=1,
w6=s3,
w7=1;
m81 m1(out,w0,w1,w2,w3,w4,w5,w6,w7,s0,s1,s2);
endmodule
module m81(output out, input D0, D1, D2, D3, D4, D5, D6, D7, S0, S1, S2);
assign out = (D0 & ~S2 & ~S1 & ~S0) | (D1 & ~S2 & ~S1 & S0) | (D2 & ~S2 & S1 &
~S0) + (D3 & ~S2 & S1 & S0) + (D4 & S2 & ~S1 & ~S0) + (D5 & S2 & ~S1 & S0) + (D6
& S2 & S1 & ~S0) + (D7 & S2 & S1 & S0);
endmodule
```

TEST BENCH

```
module test;
reg D0,D1,D2,D3,D4,D5,D6,D7,S0,S1,S2;
wire out;
m81 ins(out,D0,D1,D2,D3,D4,D5,D6,D7,S0,S1,S2);
initial begin
{D0,D1,D2,D3,D4,D5,D6,D7}={8'b00000000};
repeat(2**8)begin
{D0,D1,D2,D3,D4,D5,D6,D7}={D0,D1,D2,D3,D4,D5,D6,D7}+{8'b00000001};
S0=$ranodm();
S1=$ranodm();
S2=$ranodm();
#10;
end
end
endmodule
```

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

```
module twofourdecoder(input a,b,e,output o1,o2,o3,o4);
assign o1=~e&~a&~b;
assign o2=~e&~a&b;
assign o3=~e&a&~b;
assign o4=~e&a&b;
endmodule

module conv(input a,b,c,d,e,s1,s0, output o);
wire o1,o2,o3,o4;
wire o_1,o_2,o_3,o_4;
twofourdecoder ins(s1,s0,e,o1,o2,o3,o4);
assign o_1=o1?a:1'b0;
assign o_2=o2?b:1'b0;
assign o_3=o3?c:1'b0;
assign o_4=o4?d:1'b0;
assign o=o_1|o_2|o_3|o_4;
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 diff,borrow,input a,b);
assign diff=a^b;
assign borrow=~a&b;
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 diff,borrow,input x,y,z);
assign diff=(~x&~y&z)|(~x&y&~z)|(x&~y&~z)|(x&y&z),
      borrow=(x&~y)|(x&~z)|(y&z);
                  endmodule
```

<u>TEST BENCH</u>

```
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
```

<u>Q2: 4-bit arithmetic logic unit (ALU) performing the following operations:</u>
<u>i.</u>
<u>AND ii. OR iii. Add</u>
<u>iv. Subtract v. Less (If the first operand is less than the second</u>

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
```