ASSIGNMENT 2 DATA MODELLING

1.

1.

```
module assg2(output [3:0] Y, input [1:0] A, input din);
assign Y[0] = din & (~A[0]) & (~A[1]);
assign Y[1] = din & (~A[1]) & A[0];
assign Y[2] = din & A[1] & (~A[0]);
assign Y[3] = din & A[1] & A[0];
endmodule
end
endmodule
module assg2_tb;
 wire [3:0] Y;
 reg [1:0] A;
 reg din;
 integer i;
 assg2 instance0(Y,A,din);
 initial begin
  din=1;
    $monitor("%t| Din = %d| A[1] = %d| A[0] = %d| Y[0] = %d| Y[1] = %d| Y[2] = %d| Y[3] = %d",
            $time, din, A[1], A[0], Y[0], Y[1], Y[2], Y[3]);
  for (i = 0; i < 4; i = i+1)
   begin
   #10 {A[1],A[0]}=i;
   end
   end
endmodule
```

2.

```
module assg2(input in,input [0:2] s,output [0:7] d);
  assign d[0]=(in & ~s[2] & ~s[1] & ~s[0]),
   d[1]=(in & ~s[2] & ~s[1] & s[0]),
   d[2]=(in & ~s[2] & s[1] & ~s[0]),
   d[3]=(in & ~s[2] & s[1] & s[0]),
   d[4]=(in & s[2] & ~s[1] & ~s[0]),
   d[5]=(in & s[2] & ~s[1] & s[0]),
   d[6]=(in & s[2] & s[1] & ~s[0]),
   d[7]=(in & s[2] & s[1] & s[0]);
endmodule

module assg2_tb;
 reg in;
 reg [0:2] s;
 wire [0:7] d;
 integer i;
 assg2 instance0(in,s,d);
 initial begin
  in=1;
  $monitor("%t| in = %d| s[0] = %d | s[1] = %d| s[0] = %d| d[0] = %d| d[1] = %d| d[2] = %d| d[3] = %d  | d[4] = %d|
d[5] = %d| d[6] = %d| d[7] = %d",$time,in, s[2],s[1], s[0], d[0], d[1], d[2], d[3],d[4],d[5],d[6],d[7]);
  for(i=0;i<8;i=i+1)
   begin
```

```
   {s[2],s[1],s[0]}=i;
   end
 end
endmodule
```

3.

```
module assg2(in,d, en);
input [2:0]  in;
input en;
output [7:0] d;
  assign d[0]=~(en & ~in[0] &~in[1] & ~in[2]),
   d[1]=~(en & ~in[0] & ~in[1] & in[2]),
   d[2]=~(en & ~in[0] & in[1] & ~in[2]),
   d[3]=~(en & ~in[0] & in[1] &  in[2]),
   d[4]=~(en & in[0] & ~in[1] & ~in[2]),
   d[5]=~(en & in[0] & in[1] &  in[2]),
   d[6]=~(en & in[0] & in[1] & ~in[2]),
   d[7]=~(en & in[0] & in[1] & in[2]);
endmodule

module assg2_tb;
wire [7:0] out;
reg en;
reg [2:0] in;
integer i;

assg2 instance0(in,out,en);

initial begin
 $monitor( "en=%b, in=%d, out=%b ", en, in, out);
   for ( i=0; i<16; i=i+1)
       begin
          {en,in}  = i;
          #10;
       end
end
endmodule
```

4.

```
module decoder38(in,d, en);
input [2:0]  in;
input en;
output [7:0] d;
  assign d[0]=(en & ~in[0] &~in[1] & ~in[2]),
   d[1]=(en & ~in[0] & ~in[1] & in[2]),
   d[2]=(en & ~in[0] & in[1] & ~in[2]),
   d[3]=(en & ~in[0] & in[1] &  in[2]),
   d[4]=(en & in[0] & ~in[1] & ~in[2]),
   d[5]=(en & in[0] & in[1] &  in[2]),
   d[6]=(en & in[0] & in[1] & ~in[2]),
   d[7]=(en & in[0] & in[1] & in[2]);
endmodule
```

```
module assg2(in,d);
input [3:0] in;
output [15:0] d;
decoder38 d1(in[2:0],d[7:0],~in[3]);
decoder38 d2(in[2:0],d[15:8],in[3]);
endmodule

module assg2_tb;
wire [15:0] out;
reg [3:0] in;
integer i;

assg2 instance0(in,out);

initial begin
 $monitor( "en=%b, in=%d, out=%b ", en, in, out);
   for ( i=0; i<16; i=i+1)
       begin
          {in}  = i;
          #10;
       end
end
endmodule
```

5.

```
module full_adder(output s,c_out,input a,b,c_in);
wire f1,f2;
assign s=((a^b)^c_in);
assign f1=(a&b);
assign f2=(c_in&(a^b));
assign c_out=(f1|f2);
endmodule

module bit41multiply(input [3:0] a,input b,output [3:0] c);
assign c[0]=(b & a[0]);
assign c[1]=(b & a[1]);
assign c[2]=(b & a[2]);
assign c[3]=(b & a[3]);
endmodule

module bit7add(output [6:0] Sum,output Cout,input [6:0] A,B,input Cin);
wire c1,c2,c3,c4,c5,c6;
full_adder FA1(Sum[0],c1,A[0],B[0],Cin),
FA2(Sum[1],c2,A[1],B[1],c1),
FA3(Sum[2],c3,A[2],B[2],c2),
FA4(Sum[3],c4,A[3],B[3],c3),
FA5(Sum[4],c5,A[4],B[4],c4),
FA6(Sum[5],c6,A[5],B[5],c5),
FA7(Sum[6],Cout,A[6],B[6],c6);
endmodule

module multiplier(input [3:0] a,b,output [6:0] product,output c);
wire [6:0] p1,p2,p3,p4;
assign p1[6]=0,p1[5]=0,p1[4]=0,p2[6]=0,p2[5]=0,p2[0]=0,p3[6]=0,p3[1]=0,p3[0]=0,p4[2]=0,p4[1]=0,p4[0]=0;
```

```
bit41multiply b1(a[3:0],b[0],p1[3:0]);
bit41multiply b2(a[3:0],b[1],p2[4:1]);
bit41multiply b3(a[3:0],b[2],p3[5:2]);
bit41multiply b4(a[3:0],b[3],p4[6:3]);
wire [6:0] w1,w2;
wire w3,w4;
bit7add a1(w1,w3,p1,p2,1'b0);
bit7add a2(w2,w4,w1,p3,w3);
bit7add a3(product,c,p4,w2,w4);
endmodule

module multiplier_tb;
 reg [3:0] a,b;
 wire [6:0] p;
 wire c_out;
 integer i;
 multiplier Instance (a,b,p,c_out);
 initial begin
  a <= 0;
  b <= 0;
  $monitor ("a=%b b=%b product=%d%b", a, b, c_out,p);
  for (i = 0; i < 16; i = i+1)
   begin
   #10 a <= $random;
   b <= $random;
   end
 end
endmodule
```

6.

```
module decoder38(a,b,c,d);
input a,b,c;
output [7:0] d;
  assign d[0]=(~a & ~b & ~c),
   d[1]=(~a & ~b & c),
   d[2]=(~a & b & ~c),
   d[3]=(~a & b & c),
   d[4]=(a & ~b & ~c),
   d[5]=(a & ~b & c),
   d[6]=(a & b & ~c),
   d[7]=(a & b & c);
endmodule

module assg2(a,b,cin,cout,s);
input a,b,cin;
output cout,s;
wire [7:0] y;
decoder38 d1(a,b,cin,y[7:0]);
assign s=( y[1] | y[2] | y[4] | y[7] );
assign cout=( y[3] | y[5] | y[6] | y[7] );
endmodule

module assg2_tb;
```

```
wire cout,s;
reg a,b,cin;
integer i;
assg2 instance0(a,b,cin,cout,s);
initial begin
 $monitor( "en=%b, in=%d, out=%b ", en, in, out);
   for ( i=0;i<8; i=i+1)
       begin
          {a,b,cin}  = i;
          #10;
       end
end
endmodule
```

7.

BINARY TO GRAY

```
module assg2(input [7:0] bin,output [7:0] G);
assign G[7] = bin[7];
assign G[6] = bin[7] ^ bin[6];
assign G[5] = bin[6] ^ bin[5];
assign G[4] = bin[5] ^ bin[4];
assign G[3] = bin[4] ^ bin[3];
assign G[2] = bin[3] ^ bin[2];
assign G[1] = bin[2] ^ bin[1];
assign G[0] = bin[1] ^ bin[0];
endmodule

module assg2_tb;
   reg [7:0] bin;
   wire [7:0] G;
 integer i;
   assg2 instance0(bin,G);
 initial begin
     $monitor("bin=%b G=%b",G,bin);
     for( i=0;i<128; i=i+1)
       begin
          {bin}  = i;
          #10;
       end
   end
endmodule
```

GRAY TO BINARY

```
module assg2(input [7:0] G,output [7:0] bin);
assign bin[7] = G[7];
assign bin[6] = bin[7] ^ G[6] ;
assign bin[5] = bin[6] ^ G[5] ;
assign bin[4] = bin[4] ^ G[4] ;
assign bin[3] = bin[3] ^ G[3] ;
assign bin[2] = bin[2] ^ G[2] ;
assign bin[1] = bin[1] ^ G[1] ;
```

```
assign bin[0] = bin[0] ^ G[0] ;
endmodule

module assg2_tb;
  wire [7:0] bin;
  reg [7:0] G;
 integer i;
  assg2 instance0(G,bin);
 initial begin
    $monitor("bin=%b G=%b",G,bin);
    for( i=0;i<128; i=i+1)
      begin
        {G}  = i;
        #10;
      end
  end
endmodule
```

8.

```
module m81(output out, input D0, D1, D2, D3, D4, D5, D6, D7, S0, S1, S2);
assign S1bar=~S1;
assign S0bar=~S0;
assign S2bar=~S2;
assign out = (D0 & S2bar & S1bar & S0bar) | (D1 & S2bar & S1bar & S0) | (D2 & S2bar & S1 & S0bar) | (D3 & S2
bar & S1 & S0) | (D4 & S2 & S1bar & S0bar) | (D5 & S2 & S1bar & S0) | (D6 & S2 & S1 & S0bar) | (D7 & S2 & S
1 & S0);
endmodule

module quad(input w,x,y,z,output f);
m81 m1(f,w,~w,1'b0,~w,1'b0,1'b1,w,1'b1,z,y,x);
endmodule

module quad_tb;
 reg w,x,y,z;
 wire f;
 integer i;
 quad Instance (w,x,y,z,f);
 initial begin
 w <= 0;
 x <= 0;
 y <= 0;
 z <= 0;
  $monitor ("w=%d x=%d y=%d z=%d f=%d",w,x,y,z,f);
  for (i = 0; i <16; i = i+1)
   begin
   #10 {w,x,y,z}=i;
   end
 end
endmodule
```

9.

```
module m81(output out, input D0, D1, D2, D3, D4, D5, D6, D7, S0, S1, S2);
assign S1bar=~S1;
```

```
assign S0bar=~S0;
assign S2bar=~S2;
assign out = (D0 & S2bar & S1bar & S0bar) | (D1 & S2bar & S1bar & S0) | (D2 & S2bar & S1 & S0bar) | (D3 & S2
bar & S1 & S0) | (D4 & S2 & S1bar & S0bar) | (D5 & S2 & S1bar & S0) | (D6 & S2 & S1 & S0bar) | (D7 & S2 & S
1 & S0);
endmodule

module quad(input w,x,y,z,output f);
m81 m1(f,w,~w,1'b0,~w,1'b0,1'b1,w,1'b1,x,y,z);
endmodule

module quad_tb;
 reg w,x,y,z;
 wire f;
 integer i;
 quad Instance (w,x,y,z,f);
 initial begin
 w <= 0;
 x <= 0;
 y <= 0;
 z <= 0;
  $monitor ("w=%d x=%d y=%d z=%d f=%d",w,x,y,z,f);
  for (i = 0; i <16; i = i+1)
   begin
   #10 {w,x,y,z}=i;
   end
 end
endmodule
10.

module decoder(input [0:1] s,output [0:3] d);
  assign d[0]=(~s[1] & ~s[0]),
   d[1]=(~s[1] & s[0]),
   d[2]=(s[1] & ~s[0]),
   d[3]=(s[1] & s[0]);
endmodule

module buffer(input en,a,output b);
  assign b= en ? a : 1'bz ;
endmodule

module multiplier(input [0:3] a,input [0:1] s,output f);
 wire [3:0] w,t;
 decoder d1(s,w);
 buffer b1(w[0],a[0],t[0]);
 buffer b2(w[1],a[1],t[1]);
 buffer b3(w[2],a[2],t[2]);
 buffer b4(w[3],a[3],t[3]);
 assign f=t[s];
endmodule

module multiplier_tb;
 reg [0:3] a;
 reg [0:1] s;
 wire f;
```

```
 integer i;
 multiplier Instance (a,s,f);
 initial begin
  a <= 0;
  s <= 0;
  $monitor ("a=%b s=%b f=%d", a,s,f);
  for (i = 0; i < 16; i = i+1)
   begin
   #10 a <= $random;
   s <= $random;
   end
 end
endmodule
```

11.

```
module assg2(output D, B, input X, Y);
assign D = X ^ Y;
assign B = ~X & Y;
endmodule

module assg2_tb;
wire D, B;
reg X, Y;
integer i;
assg2 Instance0(D, B, X, Y);
initial begin
 $monitor( "D=%D | B=%d | X=%d | Y=%b ",D,B,X,Y);
   for ( i=0;i<4; i=i+1)
       begin
          {X,Y}  = i;
          #10;
       end
end
endmodule
```

12.

```
module full_adder(output s,c_out,input a,b,c_in);
wire f1,f2;
assign s=((a^b)^c_in);
assign f1=(a&b);
assign f2=(c_in&(a^b));
assign c_out=(f1|f2);
endmodule

module assg2(output [3:0] d,output c_out,input [3:0] A,B);
wire c1,c2,c3;
full_adder FA1(d[0],c1,A[0],~B[0],1'b1),
FA2(d[1],c2,A[1],~B[1],c1),
FA3(d[2],c3,A[2],~B[2],c2),
FA4(d[3],c_out,A[3],~B[3],c3);
endmodule
```

```
module assg2_tb;
reg [3:0] a,b;
wire c_out;
wire [3:0] d;
integer i;
assg2 Instance (d,c_out,a,b);
initial begin
a <= 0;
b <= 0;
$monitor ("a=0x%0h b=0x%0h c_out=0x%0h sum=0x%0h", a, b,c_out,d);
for (i = 0; i < 10; i = i+1) begin
#10 a <= $random;
b <= $random;
end
end
endmodule
```

13.

1 BIT ADDER USING FULL ADDER

```
module fulladder(output s,c_out,input a,b,c_in);
wire f1,f2;
assign s=((a^b)^c_in);
assign f1=(a&b);
assign f2=(c_in&(a^b));
assign c_out=(f1|f2);
endmodule

module alu(input a,b,output c,s);
fulladder f1(s,c,a,b,1'b0);
endmodule

module alu_tb;
 reg a,b;
 wire c,s;
 integer i;
 alu instance0(a,b,c,s);
 initial begin
   a <= 0;
   b <= 0;
   for(i=0;i<4;i=i+1)
   begin
   {a,b}=i;
   #10;
   end
 end
endmodule
```

1 BIT ADDER USING 2*1 MUX

```
module mux21(input a,b,s,output f);
assign f = (~s & a ) | ( s & b ) ;
endmodule
```

```
module alu(input a,b,output c,s);
mux21 m1(a,~a,b,s);
mux21 m2(1'b0,a,b,c);
endmodule

module alu_tb;
 reg a,b;
 wire c,s;
 integer i;
 alu instance0(a,b,c,s);
 initial begin
   a <= 0;
   b <= 0;
   for(i=0;i<4;i=i+1)
   begin
   {a,b}=i;
   #100;
   end
 end
endmodule
```

1 BIT ADDER USING INVERTER

```
module alu(input a,b,output c,s);
wire f1,f2,f3,f4;
assign f1=~a;
assign f2=~b;
assign f3=~f1;
assign f4=~f2;
assign s=( f3 & f2 ) | ( f4 & f1 ) ;
assign c=(f3 & f4);
endmodule

module alu_tb;
 reg a,b;
 wire c,s;
 integer i;
 alu instance0(a,b,c,s);
 initial begin
   a <= 0;
   b <= 0;
   for(i=0;i<4;i=i+1)
   begin
   {a,b}=i;
   #100;
   end
 end
endmodule
```

2.

```
module alu(input [3:0] A,B,input [2:0] ALU_Sel,output reg [3:0] ALU_Result,output CarryOut);
    wire [4:0] tmp;
    assign tmp = {1'b0,A} + {1'b0,B};
```

```
    assign CarryOut = tmp[4];
    always @(*)
    begin
       case(ALU_Sel)
         3'b001: //  Logical and
          ALU_Result = A & B;
         3'b010: //  Logical or
          ALU_Result = A | B;
    3'b011: // Addition
          ALU_Result = A + B ;
         3'b100: // Subtraction
          ALU_Result = A - B ;
         3'b101: // Lesser comparison
          ALU_Result = (A<B) ? 4'd1 : 4'd0 ;
       endcase
    end
endmodule

module alu_tb;
 reg[3:0] A,B;
 reg[2:0] ALU_Sel;
 wire[3:0] ALU_Out;
 wire CarryOut;
 integer i;
 alu instance0(A,B, ALU_Sel, ALU_Out, CarryOut);
    initial begin
      ALU_Sel = 3'h0;
      for (i=0;i<7;i=i+1)
      begin
       ALU_Sel = ALU_Sel + 3'h1;
   A <= $random;
   B <= $random;
       #10;
      end

    end
endmodule
```