ASSIGNMENT 2 BEHAVIORAL MODELLING

1.

1.

```
module assg2(output reg [3:0] Y, input [1:0] A, input din);
always@(Y,A)
   begin
    case (A)
  2'b00 : begin
     Y[0] = din; Y[3:1] = 0;
      end
  2'b01 : begin
     Y[1] = din; Y[0]   = 0;
      end
  2'b10 : begin
     Y[2] = din; Y[1:0] = 0;
      end
  2'b11 : begin
     Y[3] = din; Y[2:0] = 0;
      end
  endcase
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 i,input [0:2] s,output reg [0:7] out);
  always @ (i or s[0] or s[1] or s[2])
case ({s[2],s[1],s[0]})
    0: out[0] = i;
    1: out[1] = i;
    2: out[2] = i;
    3: out[3] = i;
    4: out[4] = i;
    5: out[5] = i;
    6: out[6] = i;
    7: out[7] = i;
```

```
    default: out = 8'bxxxxxxx;
endcase
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,out, en);
input [2:0]  in;
input en;
output reg [7:0] out;
 always @( in or en)
 begin

     if (en)
       begin
         out=8'd0;
         case (in)
             3'b000: out[0]=1'b0;
             3'b001: out[1]=1'b0;
             3'b010: out[2]=1'b0;
             3'b011: out[3]=1'b0;
             3'b100: out[4]=1'b0;
             3'b101: out[5]=1'b0;
             3'b110: out[6]=1'b0;
             3'b111: out[7]=1'b0;
             default: out=8'd1;
         endcase
     end
else
out=8'd1;
end
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(input [2:0] in,output reg [7:0] out,input en);
 always @( in or en)
 begin
     if (en)
       begin
         out=8'd0;
         case (in)
            3'b000: out[0]=1'b1;
            3'b001: out[1]=1'b1;
            3'b010: out[2]=1'b1;
            3'b011: out[3]=1'b1;
            3'b100: out[4]=1'b1;
            3'b101: out[5]=1'b1;
            3'b110: out[6]=1'b1;
            3'b111: out[7]=1'b1;
            default: out=8'd0;
         endcase
     end
else
out=8'd0;
end
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 reg s,c_out,input a,b,c_in);
always @ (a or b or c_in) begin
{c_out,s} = a + b + c_in;
end
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(input a,b,c,output reg [7:0] out);
 always @(a or b or c)
 begin
        out=8'd0;
        case ({a,b,c})
            3'b000: out[0]=1'b1;
            3'b001: out[1]=1'b1;
            3'b010: out[2]=1'b1;
            3'b011: out[3]=1'b1;
            3'b100: out[4]=1'b1;
            3'b101: out[5]=1'b1;
            3'b110: out[6]=1'b1;
            3'b111: out[7]=1'b1;
            default: out=8'd0;
        endcase
  end
endmodule

module assg2(a,b,cin,cout,s);
input a,b,cin;
output reg cout,s;
wire [7:0] y;
decoder38 d1(a,b,cin,y[7:0]);
always@(y[1] or y[2] or y[4] or y[7])
  begin
    if(y[1]==1'b0 & y[2]==1'b0 & y[4]==1'b0 & y[7]==1'b0 )
    begin
    s=1'b0;
  end
 else
   begin
     s=1'b1;
   end
  end
always@(y[3] or y[5] or y[6] or y[7])
  begin
    if(y[3]==1'b0 & y[5]==1'b0 & y[6]==1'b0 & y[7]==1'b0 )
    begin
    cout=1'b0;
  end
 else
   begin
```

```
    cout=1'b1;
   end
  end
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 reg [7:0] G);
always@(bin[7])
  begin
    if(bin[7]==1'b0)
    begin
    G[7]=1'b0;
  end
 else
   begin
     G[7]=1'b1;
   end
  end
always@(bin[7] or bin[6])
  begin
    if(bin[7]==bin[6])
    begin
    G[6]=1'b0;
  end
 else
   begin
     G[6]=1'b1;
   end
  end
always@(bin[6] or bin[5])
  begin
    if(bin[5]==bin[6])
    begin
    G[5]=1'b0;
  end
 else
```

```
   begin
     G[5]=1'b1;
   end
  end
always@(bin[5] or bin[4])
  begin
    if(bin[5]==bin[4])
    begin
    G[4]=1'b0;
  end
 else
   begin
     G[4]=1'b1;
   end
  end
always@(bin[4] or bin[3])
  begin
    if(bin[3]==bin[4])
    begin
    G[3]=1'b0;
  end
 else
   begin
     G[3]=1'b1;
   end
  end
always@(bin[3] or bin[2])
  begin
    if(bin[3]==bin[2])
    begin
    G[2]=1'b0;
  end
 else
   begin
     G[2]=1'b1;
   end
  end
always@(bin[2] or bin[1])
  begin
    if(bin[2]==bin[1])
    begin
    G[1]=1'b0;
  end
 else
   begin
     G[1]=1'b1;
   end
  end
 always@(bin[1] or bin[0])
  begin
    if(bin[1]==bin[0])
    begin
    G[0]=1'b0;
  end
 else
```

```
   begin
     G[0]=1'b1;
   end
  end
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] bin,output reg [7:0] G);
always@(bin[7])
  begin
    if(bin[7]==1'b0)
    begin
    G[7]=1'b0;
  end
 else
   begin
     G[7]=1'b1;
   end
  end
always@(G[7] or bin[6])
  begin
    if(G[7]==bin[6])
    begin
    G[6]=1'b0;
  end
 else
   begin
     G[6]=1'b1;
   end
  end
always@(G[6] or bin[5])
  begin
    if(bin[5]==G[6])
    begin
    G[5]=1'b0;
  end
 else
   begin
     G[5]=1'b1;
```

```
   end
  end
always@(G[5] or bin[4])
  begin
    if(G[5]==bin[4])
    begin
    G[4]=1'b0;
  end
 else
   begin
     G[4]=1'b1;
   end
  end
always@(G[4] or bin[3])
  begin
    if(bin[3]==G[4])
    begin
    G[3]=1'b0;
  end
 else
   begin
     G[3]=1'b1;
   end
  end
always@(G[3] or bin[2])
  begin
    if(G[3]==bin[2])
    begin
    G[2]=1'b0;
  end
 else
   begin
     G[2]=1'b1;
   end
  end
always@(G[2] or bin[1])
  begin
    if(G[2]==bin[1])
    begin
    G[1]=1'b0;
  end
 else
   begin
     G[1]=1'b1;
   end
  end
 always@(G[1] or bin[0])
  begin
    if(G[1]==bin[0])
    begin
    G[0]=1'b0;
  end
 else
   begin
     G[0]=1'b1;
```

```
   end
  end
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 quad(input [0:3] a,b,input e,s,output reg [0:3] f);
 always@(*)
 begin
   if(~e)
     if(s)
     f=b;
   else
     f=a;
  else
    f=4'b0000;
 end
endmodule

module quad_tb;
 reg [0:3] a,b;
 reg e,s;
 wire [0:3] f;
 integer i;
 quad Instance (a,b,e,s,f);
 initial begin
  a <= 0;
  b <= 0;
  e <= 0;
  s <= 0;
  $monitor ("a=%b b=%b e=%d s=%d f=%b", a,b,e,s,f);
  for (i = 0; i < 16; i = i+1)
   begin
   #10 a <= $random;
   b <= $random;
   e <= $random;
   s <= $random;
   end
 end
endmodule
```

9.

```
module m81(output reg out, input D0, D1, D2, D3, D4, D5, D6, D7, S0, S1, S2);
 always@(*)
 begin
  case({S0,S1,S2})
  3'b000: out=D0;
  3'b001: out=D1;
  3'b010: out=D2;
  3'b011: out=D3;
  3'b100: out=D4;
  3'b101: out=D5;
  3'b110: out=D6;
  3'b111: out=D7;
  default: out=1'b0;
  endcase
 end
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] in,output reg [0:3] out);
 always @( in )
 begin
        case (in)
           2'b00: out[0]=1'b1;
           2'b01: out[1]=1'b1;
           2'b10: out[2]=1'b1;
           2'b11: out[3]=1'b1;
           default: out=4'd0;
        endcase
  end
```

```
endmodule

module buffer(input en,a,output reg b);
 always@(*)
   begin
   if(en)
   b=a;
   else
   b=1'bz;
  end
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 reg D, B, input X, Y);
always@(X or Y)
  begin
    if(X==Y)
    begin
    D=1'b0;
  end
 else
   begin
     D=1'b1;
   end
  end
```

```
always@(X or Y)
  begin
    if(~X==1'b1 & Y==1'b1 )
    begin
    B=1'b1;
  end
 else
   begin
     B=1'b0;
   end
  end
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 reg s,c_out,input a,b,c_in);
always @ (a or b or c_in) begin
{c_out,s} = a + b + c_in;
end
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 reg s,c_out,input a,b,c_in);
always @ (a or b or c_in) begin
{c_out,s} = a + b + c_in;
end
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 reg f);
always@(*)
 case(s)
 1'b0: f=a;
 1'b1: f=b;
 endcase
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 reg c,s);
always@(a or b)
 begin
  if(a==b)
    s=1'b0;
  else
    s=1'b1;
 end
always@(a or b)
  begin
    if(a==1'b1 && b==1'b1)
    c=1'b1;
  else
    c=1'b0;
 end
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
           begin
               ALU_Result = A & B;
           end
          3'b010: //  Logical or
           begin
               ALU_Result = A | B;
           end
    3'b011: // Addition
           begin
               ALU_Result = A + B;
           end
          3'b100: // Subtraction
           begin
               ALU_Result = A - B;
           end
          3'b101: // Lesser comparison
           begin
               ALU_Result = A < B;
           end
        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
```