Assignment_2_Extra

==========

Name: Ahmed Nour

Sheet ID: 16

Email: ahmedmohamednoour@gmail.com

==========

Q1)

Models Code:

```
module Gray_encoder (B_g, A_g);
    input [2:0] A_g;
    reg [2:0] gray_T;
    output reg [6:0] B_g;

    always @(*) begin
        case(A_g)
            3'd0 : gray_T = 3'b000;
            3'd1 : gray_T = 3'b001;
            3'd2 : gray_T = 3'b011;
            3'd3 : gray_T = 3'b010;
            3'd4 : gray_T = 3'b110;
            3'd5 : gray_T = 3'b111;
            3'd6 : gray_T = 3'b101;
            3'd7 : gray_T = 3'b100;
            default: gray_T = 3'b000;
        endcase
        B_g = {4'b0 , gray_T};
    end
endmodule
```

```
module Onehot_encoder (B_oh, A_oh);
    input [2:0] A_oh;
    output reg [6:0] B_oh;

    always @(*) begin
        B_oh = 0;
        case(A_oh)
        3'd0: B_oh = 7'b000_0000;
        default: B_oh[A_oh - 1] = B_oh[A_oh - 1] | 1;
        endcase
    end
endmodule
```

```
module Gray_onehot_encoder (B, A);
    parameter USE_GRAY =1;

    input [2:0] A;
    output [6:0] B;

    generate
        if (USE_GRAY == 1) begin
            Gray_encoder G_encoder1 (B, A);
            //this encoder get the 3-bit code and place it in the 3 least bits in B, as B is 7 bits in the specs.
        end
        else begin
            Onehot_encoder Oh_encoder2 (B, A);
        end
    endgenerate

endmodule
```

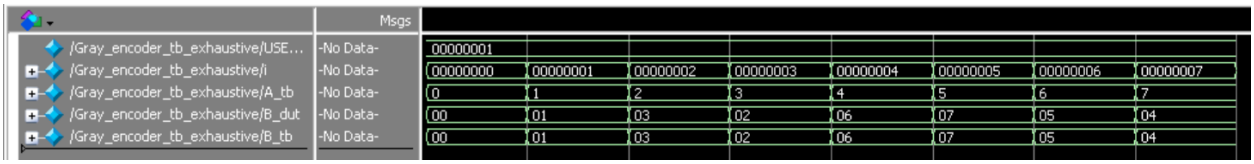
Testbench Codes:

```
module Gray_encoder_tb_exhaustive();
    parameter USE_GRAY_tb = 1;
    reg [2:0] A_tb;
    reg [2:0] gray_tb;
    reg [6:0] B_tb;
    wire [6:0] B_dut;

    Gray_onehot_encoder #(USE_GRAY_tb) DUT (B_dut, A_tb);

    integer i;
    initial begin
        for (i=0; i<8; i=i+1) begin
            A_tb = i;
            case(A_tb)
            3'd0 : gray_tb = 3'b000;
            3'd1 : gray_tb = 3'b001;
            3'd2 : gray_tb = 3'b011;
            3'd3 : gray_tb = 3'b010;
            3'd4 : gray_tb = 3'b110;
            3'd5 : gray_tb = 3'b111;
            3'd6 : gray_tb = 3'b101;
            3'd7 : gray_tb = 3'b100;
            endcase
            B_tb = {4'b0 , gray_tb};
            #10;
            $display("Value of golden model = %b , Value of DUT = %b", B_tb, B_dut);
        end
    end

endmodule
```

```
module Onehot_encoder_tb_exhaustive();
    parameter USE_GRAY_tb = 0;
    reg [2:0] A_tb;
    reg [2:0] gray_tb;
    reg [6:0] B_tb;
    wire [6:0] B_dut;

    Gray_onehot_encoder #(USE_GRAY_tb) DUT (B_dut, A_tb);

    integer i;
    initial begin
        for (i=0; i<8; i=i+1) begin
            A_tb = i;
            case(A_tb)
            3'd0 : B_tb = 7'b000_0000;
            3'd1 : B_tb = 7'b000_0001;
            3'd2 : B_tb = 7'b000_0010;
            3'd3 : B_tb = 7'b000_0100;
            3'd4 : B_tb = 7'b000_1000;
            3'd5 : B_tb = 7'b001_0000;
            3'd6 : B_tb = 7'b010_0000;
            3'd7 : B_tb = 7'b100_0000;
            endcase
            #10;
            $display("Value of golden model = %b , Value of DUT = %b", B_tb, B_dut);
        end
    end

endmodule
```

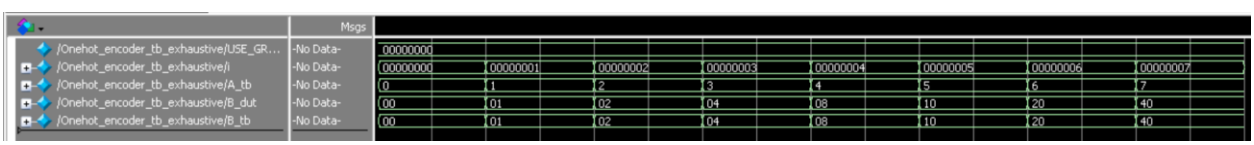
Simulation:

Grey Encoder test, Successful.




```
VSIM 7> run -all
# Value of golden model = 0000000 , Value of DUT = 0000000
# Value of golden model = 0000001 , Value of DUT = 0000001
# Value of golden model = 0000011 , Value of DUT = 0000011
# Value of golden model = 0000010 , Value of DUT = 0000010
# Value of golden model = 0000110 , Value of DUT = 0000110
# Value of golden model = 0000111 , Value of DUT = 0000111
# Value of golden model = 0000101 , Value of DUT = 0000101
# Value of golden model = 0000100 , Value of DUT = 0000100
```

One-hot Encoder test, Successful.




```
VSIM 19> run -all
# Value of golden model = 0000000 , Value of DUT = 0000000
# Value of golden model = 0000001 , Value of DUT = 0000001
# Value of golden model = 0000010 , Value of DUT = 0000010
# Value of golden model = 0000100 , Value of DUT = 0000100
# Value of golden model = 0001000 , Value of DUT = 0001000
# Value of golden model = 0010000 , Value of DUT = 0010000
# Value of golden model = 0100000 , Value of DUT = 0100000
# Value of golden model = 1000000 , Value of DUT = 1000000
```

Q2)

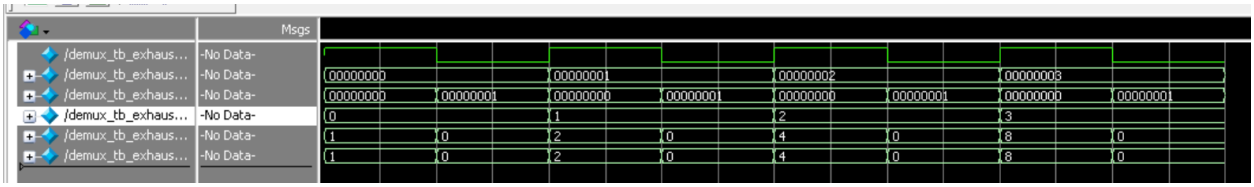
Code:

```
module demux14 (y, s, d);
input d;
input [1:0] s;
output reg [3:0] y;

always @(*) begin
    y = 0;
    case(s)
        2'b00: y[0] = d;
        2'b01: y[1] = d;
        2'b10: y[2] = d;
        default: y[3] = d;
    endcase
end
endmodule
```

Testbench:

```
module demux_tb_exhaustive_self_checking ();
reg d_tb;
reg [1:0] s_tb;
reg [3:0] y_exp, temp1;
wire [3:0] y_dut;

demux14 DUT (y_dut, s_tb, d_tb);


integer i, j;
initial begin
    temp1 = 2'b01;
    for(i = 0; i < 4; i=i+1)begin
        s_tb = i;
        for(j = 0; j < 2; j=j+1)begin
            d_tb = temp1[j];
            if(s_tb == 0)
                y_exp = {3'b000, d_tb};
            else if(s_tb == 1)
                y_exp = {2'b00, d_tb, 1'b0};
            else if(s_tb == 2)
                y_exp = {1'b0, d_tb, 2'b00};
            else
                y_exp = {d_tb, 3'b000};
            #10;
            if(y_dut !== y_exp)begin
              $display("Error at %b in the demux", s_tb);
              $stop;
            end
        end
    end
    $display("No errors found");

end

endmodule
```

Simulation:




```
# .main_pane.wave.interior.cs.body.pw.wf
add wave -position insertpoint  \
sim:/demux_tb_exhaustive_self_checking/d_tb \
sim:/demux_tb_exhaustive_self_checking/i \
sim:/demux_tb_exhaustive_self_checking/j \
sim:/demux_tb_exhaustive_self_checking/s_tb \
sim:/demux_tb_exhaustive_self_checking/y_dut \
sim:/demux_tb_exhaustive_self_checking/y_exp
VSIM 246> run -all
# No errors found
```