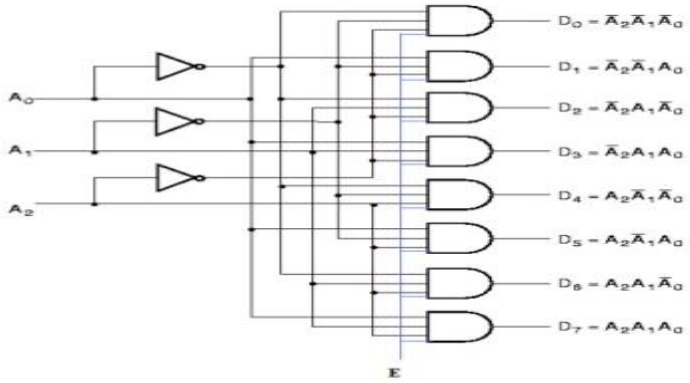
Assignment4

Q1. 3 to 8 line decoder with enable (Gate-Level)



Verilog code

```
module decoder3to8(Data_in,Data_out,E);
input [2:0] Data_in;
input E;
output [7:0] Data_out;
NOT N0(A0N,Data_in[1]);
NOT N1(A1N,Data_in[2]);
NOT N2(A2N,Data_in[3]);
AND A0(Data_out[1],A0N,A1N,A2N,E);
AND A1(Data_out[2],Data_in[1],A1N,A2N,E);
AND A2(Data_out[3],A0N, Data_in[2],A2N,E);
AND A3(Data_out[4], Data_in[1], Data_in[2],A2N,E);
AND A4(Data_out[5],A0N,A1N, Data_in[3],E);
AND A5(Data_out[6], Data_in[1],A1N, Data_in[3],E);
AND A6(Data_out[7],A0N, Data_in[2],Data_in[3],E);
AND A7(Data_out[8], Data_in[1], Data_in[2], Data_in[3],E);
endmodule
```

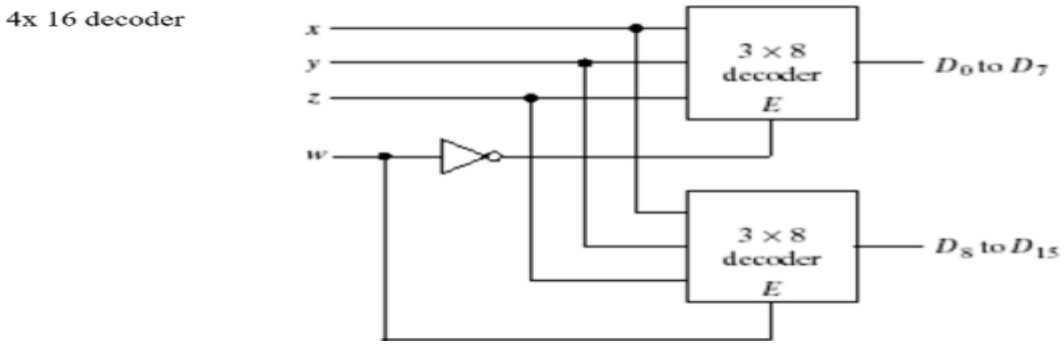
Testbench

```
module tb_decoder;
reg [2:0] Data_in;
reg E;
wire [7:0] Data_out;
decoder3to8 uut(Data_in,Data_out,E);
initial
begin

    E=1; Data_in = 3'b000;    #100;
    E=1; Data_in = 3'b001;    #100;
    E=1; Data_in = 3'b010;    #100;
```

```
    E=1; Data_in = 3'b011;    #100;
    E=1; Data_in = 3'b100;    #100;
    E=1; Data_in = 3'b101;    #100;
    E=1; Data_in = 3'b110;    #100;
    E=1; Data_in = 3'b111;    #100;
  end
endmodule
```

Q2. 4 to 16 using 3 to 8 line decoder from Q1(Structural modelling)



Verilog code

```
module decoder4to16(Data_in,Data_out,E);
input [2:0] Data_in;
input E;
output [15:0] Data_out;
NOT N (EN,E);
decoder3to8 d1(Data_in,Data_out[0:7],E);
decoder3to8 d2(Data_in,Data_out[8:15],EN);
endmodule
```

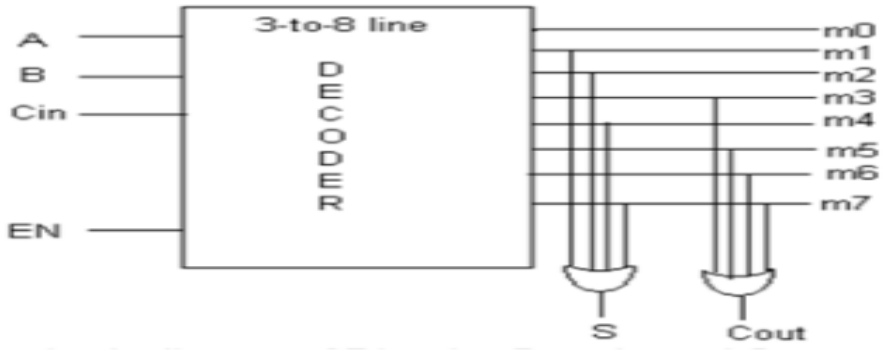
Test bench

```
module tb_decoder;
reg [2:0] Data_in;
reg E;
wire [15:0] Data_out;
decoder4to16 uut(Data_in,Data_out,E);
initial
begin

    E=1; Data_in = 3'b000;    #100; E=0; Data_in = 3'b000;    #100;
    E=1; Data_in = 3'b001;    #100; E=0; Data_in = 3'b001;    #100;
    E=1; Data_in = 3'b010;    #100; E=0; Data_in = 3'b010;    #100;
    E=1; Data_in = 3'b011;    #100; E=0; Data_in = 3'b011;    #100;
    E=1; Data_in = 3'b100;    #100; E=0; Data_in = 3'b100;    #100;
    E=1; Data_in = 3'b101;    #100; E=0; Data_in = 3'b101;    #100;
```

```
    E=1; Data_in = 3'b110;    #100; E=0; Data_in = 3'b110;    #100;
    E=1; Data_in = 3'b111;    #100; E=0; Data_in = 3'b111;    #100;
  end
endmodule
```

Q3. Full adder using 3 to 8 line decoder and OR gates (Structural modelling)



Verilog code

```
module Fulladder(Data_in,Data_out,E);
input [2:0] Data_in;
input E;
output [7:0] Data_out;
wire [3:0]sum,Cout;
wire [7:0]m;
decoder3to8 d1(Data_in, E);
OR o1(Cout, m[1], m[3], m[5], m[7]);
OR o2(Sum, m[2], m[4], m[6], m[8]);
Data_out={Cout,Sum};
endmodule
```

Testbench

```
module tb_fulladder;
reg [2:0] Data_in;
reg E;
wire [7:0] Data_out;
Fulladder f1(Data_in,Data_out,E);
initial
begin

    E=1; Data_in = 3'b000;    #100;
    E=1; Data_in = 3'b001;    #100;
    E=1; Data_in = 3'b010;    #100;
```

```
    E=1; Data_in = 3'b011;    #100;
    E=1; Data_in = 3'b100;    #100;
    E=1; Data_in = 3'b101;    #100;
    E=1; Data_in = 3'b110;    #100;
    E=1; Data_in = 3'b111;    #100;
  end
endmodule
```

Q4. 2 to 1 Multiplexer (data flow modelling)

Verilog code

```
module mux2_to_1d (out, i0, i1, s0);
output out;
input i0, i1;
input s0;
assign out=s0?i1:i0;
endmodule
```

Test bench

```
module tb_2To1d
wire out;
reg i0, i1, i2, i3;
reg s1, s0;
mux2_to_1d m1(out, i0, i1, s0);
initial
begin
i0=0; i1=1; s0=0; #100
i0=0; i1=0; s0=1; #100
i0=1; i1=0; s0=0; #100
end
endmodule
```

Q5. 4 to 1 multiplexer using 2 to 1 Multiplexer (Structural modelling)

Verilog code

```
module mux2to1 (out, i0, i1, s0);
output out;
input i0, i1;
input s0;
tri out;
bufif0 (out,i0,s0);
bufif1 (out,i1,s0);
endmodule
```

```
module mux4to1 (out, a,sel);
output out;
input [3:0]a;
input [1:0]sel;
wire mux[1:0];
mux2to1 m1(a[3],a[2],sel[0],mux[1]);
mux2to1 m2(a[1],a[4],sel[0],mux[2]);
mux2to1 m3(mux[1],mux[2],sel [1],out);
endmodule
```

Test bench

```
module tb_4to1
wire out;
reg [3:0]a;
reg [1:0]sel;
mux4to1 m1(out, a,sel);
initial
begin
a=0010; sel=00; #100
a=0010; sel=01; #100
a=0010; sel=10; #100
a=0010; sel=11; #100
end
endmodule
```

Q6. 2 to 1 Multiplexer (Behavioral modelling)

Verilog code

```
module mux2_to_1 (out, i0, i1, s0);
output out;
input i0, i1;
input s0;
reg out;
always @(*)
case (s0)
1'd0 : out = i0;
1'd1 : out = i1;
default: $display("Invalid control signal");
endcase
endmodule
```

Test bench

```
module tb_2To1
```

```
wire out;
reg i0, i1, i2, i3;
reg s1, s0;
mux2_to_1 m1(out, i0, i1, i2, i3, s1, s0);
initial
begin
i0=0; i1=1; s0=0; #100
i0=0; i1=0; s0=1; #100
i0=1; i1=0; s0=0; #100
end
endmodule
```

Q7. 4 to 1 Multiplexer (Behavioral modelling)

Verilog code

```
module mux4_to_1 (out, i0, i1, i2, i3, s1, s0);
output out;
input i0, i1, i2, i3;
input s1, s0;
reg out;
always @(s1 or s0 or i0 or i1 or i2 or i3)
case ({s1, s0})
2'd0 : out = i0;
2'd1 : out = i1;
2'd2 : out = i2;
2'd3 : out = i3;
default: $display("Invalid control signals");
endcase
endmodule
```

Test Bench

```
module tb_4To1
wire out;
reg i0, i1, i2, i3;
reg s1, s0;
mux4_to_1 m1(out, i0, i1, i2, i3, s1, s0);
initial
begin
i0=0; i1=0; i2=1; i3=1; s0=0; s1=0; #100
i0=0; i1=0; i2=1; i3=1; s0=0; s1=1; #100
i0=0; i1=0; i2=1; i3=1; s0=1; s1=0; #100
i0=0; i1=0; i2=1; i3=1; s0=1; s1=1; #100
end
endmodule
```

Instructions for implementing the above codes in ViVado

1.Create a Vivado Project using IDE

- Open Vivado by selecting Start > All Programs > Xilinx Design Tools > Vivado 2013.3 > Vivado 2013.3
- Click Create New Project to start the wizard. You will see Create A New Vivado Project dialog box. Click Next.
- Click the Browse button of the Project location field of the New Project form, browse to c:\xup\digital, and click Select.
- Enter assignment in the Project name field. Make sure that the Create Project Subdirectory box is checked. Click Next.
- Select RTL Project option in the Project Type form, and click Next.
- Select Verilog as the Target language and Simulator language in the Add Sources form.
- Click on the Add Files… button, browse to the c:\xup\digital\sources\tutorial directory, select tutorial.v, click Open, and then click Next.
- Click Next to get to the Add Constraints form.
- Click Next if the entry is already auto-populated, otherwise click on the Add Files… button, browse to the c:\xup\digital\sources\turorial directory and select tutorial.xdc, and click Open.
- In the Default Part form, using the Parts option and various drop-down fields of the Filter section, select the XC7A100TCSG324-1 part. Click Next.
- Click Finish to create the Vivado project.

2. Simulate the Design using the XSim Simulator

- Click Add Sources under the Project Manager tasks of the Flow Navigator pane
- Select the Add or Create Simulation Sources option and click Next.
- In the Add Sources Files form, click the Add Files… button.
- Browse to the c:\xup\digital\sources folder and select tutorial_tb.v and click OK. Click Finish.
- Select the Sources tab and expand the Simulation Sources group.
- Using the Windows Explorer, verify that the sim_1 directory is created at the same level as constrs_1 and sources_1 directories under the tutorial.srcs directory, and that a copy of tutorial_tb.v is placed under tutorial.srcs > sim_1 > imports > sources.
- Double-click on the tutorial_tb in the Sources pane to view its contents.
- Select Simulation Settings under the Project Manager tasks of the Flow Navigator pane.

- A Project Settings form will appear showing the Simulation properties form.
- Select the Simulation tab, and set the Simulation Run Time value to 200 ns and click OK.
- Click on Run Simulation > Run Behavioral Simulation under the Project Manager tasks of the Flow Navigator pane.