EEE 2243
Digital System Design
Semester II 2011/12

Tutorial 3

Verilog of Elementary Components

1. Write the Verilog code for the following decoders in Boolean expression approach:

   a. 2-to-4 active high

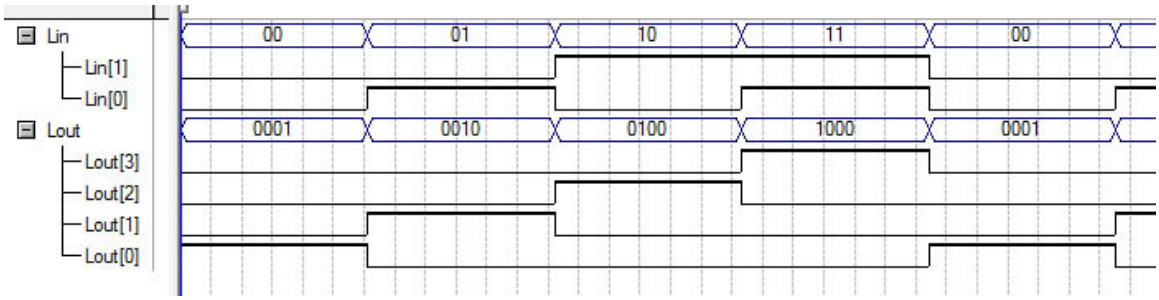
```
module Decoder_2_4_H(Lin, Lout);

input [1:0] Lin;
output [3:0] Lout;

assign Lout[0] = ~Lin[1] & ~Lin[0];
assign Lout[1] = ~Lin[1] &  Lin[0];
assign Lout[2] =  Lin[1] & ~Lin[0];
assign Lout[3] =  Lin[1] &  Lin[0];

endmodule
```

Simulation:



   b. 2-to-4 active low

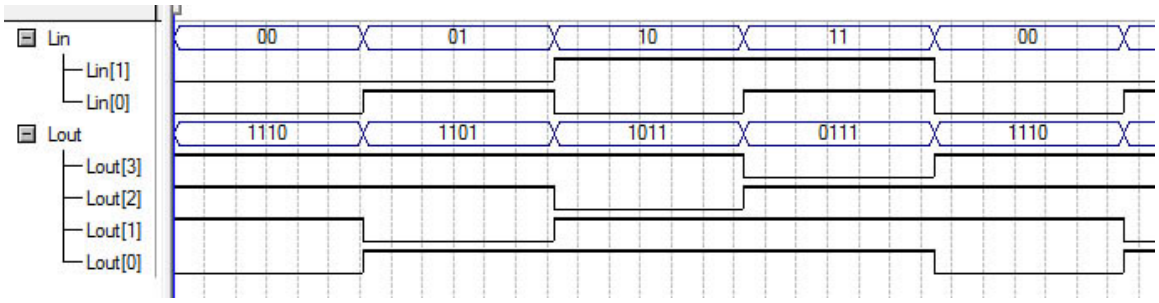
```
module Decoder_2_4_L(Lin, Lout);

input [1:0] Lin;
output [3:0] Lout;

assign Lout[0] = ~(~Lin[1] & ~Lin[0]);
assign Lout[1] = ~(~Lin[1] &  Lin[0]);
assign Lout[2] = ~( Lin[1] & ~Lin[0]);
assign Lout[3] = ~( Lin[1] &  Lin[0]);

endmodule
```

Simulation:



c. 3-to-8 active high

```
module Decoder_3_8_H(Lin, Lout);

input [2:0] Lin;
output [7:0] Lout;

assign Lout[0] = ~Lin[2] & ~Lin[1] & ~Lin[0];
assign Lout[1] = ~Lin[2] & ~Lin[1] &  Lin[0];
assign Lout[2] = ~Lin[2] &  Lin[1] & ~Lin[0];
assign Lout[3] = ~Lin[2] &  Lin[1] &  Lin[0];
assign Lout[4] =  Lin[2] & ~Lin[1] & ~Lin[0];
assign Lout[5] =  Lin[2] & ~Lin[1] &  Lin[0];
assign Lout[6] =  Lin[2] &  Lin[1] & ~Lin[0];
assign Lout[7] =  Lin[2] &  Lin[1] &  Lin[0];

endmodule
```

d. 3-to-8 active low

```
module Decoder_3_8_L(Lin, Lout);

input [2:0] Lin;
output [7:0] Lout;

assign Lout[0] = ~(~Lin[2] & ~Lin[1] & ~Lin[0]);
assign Lout[1] = ~(~Lin[2] & ~Lin[1] &  Lin[0]);
assign Lout[2] = ~(~Lin[2] &  Lin[1] & ~Lin[0]);
assign Lout[3] = ~(~Lin[2] &  Lin[1] &  Lin[0]);
assign Lout[4] = ~( Lin[2] & ~Lin[1] & ~Lin[0]);
assign Lout[5] = ~( Lin[2] & ~Lin[1] &  Lin[0]);
assign Lout[6] = ~( Lin[2] &  Lin[1] & ~Lin[0]);
assign Lout[7] = ~( Lin[2] &  Lin[1] &  Lin[0]);

endmodule
```

Simulate the 2-to-4 decoders.

2. Write the Verilog code for 4-to-1 and 8-to-1 multiplexers in Boolean expression and behavioral approach.

For Boolean approach follow the circuit in slide for chapter 3 page 14.

```
module Mux4Bool(Lin, Sel, Lout);

input [3:0] Lin;
input [1:0] Sel;
output Lout;

assign Lout = (~Sel[1] & ~Sel[0] & Lin[0]) |
              (~Sel[1] &  Sel[0] & Lin[1]) |
              ( Sel[1] & ~Sel[0] & Lin[2]) |
              ( Sel[1] &  Sel[0] & Lin[3]);

endmodule
```

```
module Mux8Bool(Lin, Sel, Lout);

input [7:0] Lin;
input [2:0] Sel;
output Lout;

assign Lout = (~Sel[2] & ~Sel[1] & ~Sel[0] & Lin[0]) |
              (~Sel[2] & ~Sel[1] &  Sel[0] & Lin[1]) |
              (~Sel[2] &  Sel[1] & ~Sel[0] & Lin[2]) |
              (~Sel[2] &  Sel[1] &  Sel[0] & Lin[3]) |
              ( Sel[2] & ~Sel[1] & ~Sel[0] & Lin[4]) |
              ( Sel[2] & ~Sel[1] &  Sel[0] & Lin[5]) |
              ( Sel[2] &  Sel[1] & ~Sel[0] & Lin[6]) |
              ( Sel[2] &  Sel[1] &  Sel[0] & Lin[7]);

endmodule
```

```
module Mux4Beha(Lin, Sel, Lout);

input [3:0] Lin;
input [1:0] Sel;
output Lout;
reg Lout

always@(*)
begin
   case (Sel)
      0: Lout <= Lin[0];
      1: Lout <= Lin[1];
      2: Lout <= Lin[2];
      3: Lout <= Lin[3];
   endcase
end

endmodule
```

```
module Mux8Beha(Lin, Sel, Lout);

input [7:0] Lin;
input [2:0] Sel;
output Lout;

always@(*)
begin
   case (Sel)
      0: Lout <= Lin[0];
      1: Lout <= Lin[1];
      2: Lout <= Lin[2];
      3: Lout <= Lin[3];
      4: Lout <= Lin[4];
      5: Lout <= Lin[5];
      6: Lout <= Lin[6];
      7: Lout <= Lin[7];
   endcase
end

endmodule
```

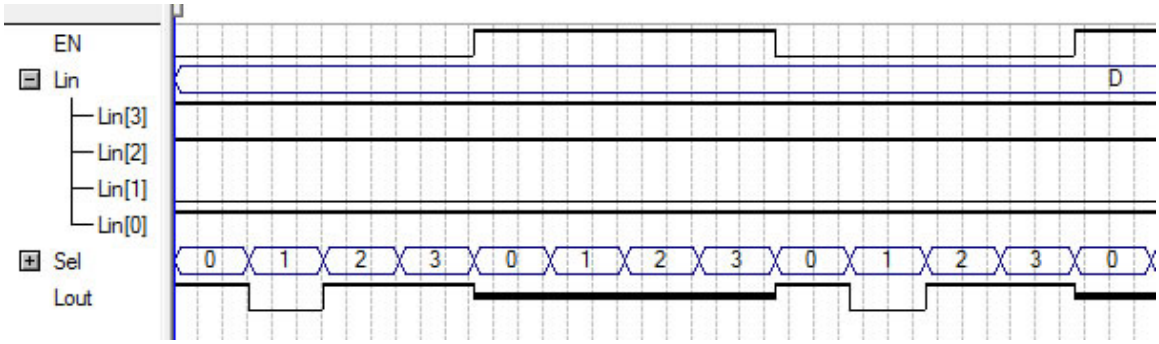
3. Repeat no. 2 but now add another input called “active low output enable.” It means the output has effect only if that input is LOW, otherwise the output will be a high-Z. Simulate at least one of them.

```
module Mux4Bool(Lin, Sel, Lout, EN);

input EN;
input [3:0] Lin;
input [1:0] Sel;
output Lout;

assign Lout = (EN) ?
              1'bz :
              ((~Sel[1] & ~Sel[0] & Lin[0]) |
               (~Sel[1] &  Sel[0] & Lin[1]) |
               ( Sel[1] & ~Sel[0] & Lin[2]) |
               ( Sel[1] &  Sel[0] & Lin[3]));

endmodule
```



```
module Mux8Bool(Lin, Sel, Lout);

input EN;
input [7:0] Lin;
input [2:0] Sel;
output Lout;

assign Lout = (EN) ?
              1'bz :
              ((~Sel[2] & ~Sel[1] & ~Sel[0] & Lin[0]) |
               (~Sel[2] & ~Sel[1] &  Sel[0] & Lin[1]) |
               (~Sel[2] &  Sel[1] & ~Sel[0] & Lin[2]) |
               (~Sel[2] &  Sel[1] &  Sel[0] & Lin[3]) |
               ( Sel[2] & ~Sel[1] & ~Sel[0] & Lin[4]) |
               ( Sel[2] & ~Sel[1] &  Sel[0] & Lin[5]) |
               ( Sel[2] &  Sel[1] & ~Sel[0] & Lin[6]) |
               ( Sel[2] &  Sel[1] &  Sel[0] & Lin[7]));

endmodule
```

```
module Mux4Beha(Lin, Sel, Lout);

input EN;
input [3:0] Lin;
input [1:0] Sel;
output Lout;
reg Lout;

always@(*)
begin
   case (Sel)
      0: Lout <= (EN) ? 1'bz : Lin[0];
      1: Lout <= (EN) ? 1'bz : Lin[1];
      2: Lout <= (EN) ? 1'bz : Lin[2];
      3: Lout <= (EN) ? 1'bz : Lin[3];
   endcase
end

endmodule
```

```
module Mux8Beha(Lin, Sel, Lout);

input EN;
input [7:0] Lin;
input [2:0] Sel;
output Lout;
reg Lout;

always@(*)
begin
   case (Sel)
      0: Lout <= (EN) ? 1'bz : Lin[0];
      1: Lout <= (EN) ? 1'bz : Lin[1];
      2: Lout <= (EN) ? 1'bz : Lin[2];
      3: Lout <= (EN) ? 1'bz : Lin[3];
      4: Lout <= (EN) ? 1'bz : Lin[4];
      5: Lout <= (EN) ? 1'bz : Lin[5];
      6: Lout <= (EN) ? 1'bz : Lin[6];
      7: Lout <= (EN) ? 1'bz : Lin[7];
   endcase
end

endmodule
```

4. Extend the decoders in no. 1 to their corresponding demultiplexers. Hint: It is easier to keep it as Boolean expression approach.

```
module Demux4(Lin, Lout, Sel);

input Lin;
input [1:0] Sel;
output [3:0] Lout;

assign Lout[0] = ~Sel[1] & ~Sel[0] & Lin;
assign Lout[1] = ~Sel[1] &  Sel[0] & Lin;
assign Lout[2] =  Sel[1] & ~Sel[0] & Lin;
assign Lout[3] =  Sel[1] &  Sel[0] & Lin;

endmodule
```

```
module Demux8(Lin, Lout, Sel);

input Lin;
input [2:0] Sel;
output [7:0] Lout;

assign Lout[0] = ~Sel[2] & ~Sel[1] & ~Sel[0] & Lin;
assign Lout[1] = ~Sel[2] & ~Sel[1] &  Sel[0] & Lin;
assign Lout[2] = ~Sel[2] &  Sel[1] & ~Sel[0] & Lin;
assign Lout[3] = ~Sel[2] &  Sel[1] &  Sel[0] & Lin;
assign Lout[4] =  Sel[2] & ~Sel[1] & ~Sel[0] & Lin;
assign Lout[5] =  Sel[2] & ~Sel[1] &  Sel[0] & Lin;
assign Lout[6] =  Sel[2] &  Sel[1] & ~Sel[0] & Lin;
assign Lout[7] =  Sel[2] &  Sel[1] &  Sel[0] & Lin;

endmodule
```

5. In both Boolean and behavioral approaches, write the Verilog code for converting D flip-flops into T and JK flip-flop.

```
module TFlipFlopBool(clk, T, Q);

input T, clk;
output Q;
reg Q;

always@(posedge clk)
begin
   Q <= (~T & Q) | (T & ~Q);
end

endmodule
```

```
module TFlipFlopBeha(clk, T, Q);

input T, clk;
output Q;
reg Q;

always@(posedge clk)
begin
   if (T == 1)
      Q <= ~Q;
   else
      Q <= Q;
end

endmodule
```

```
module JKFlipFlopBool(clk, J, K, Q);

input J, K, clk;
output Q;
reg Q;

always@(posedge clk)
begin
   Q <= (~K & Q) | (J & ~Q);
end

endmodule
```

```
module JKFlipFlopBool(clk, J, K, Q);

input J, K, clk;
output Q;
reg Q;

always@(posedge clk)
begin
   if      ((J == 0) && (K == 0)) Q <= Q;
   else if ((J == 0) && (K == 1)) Q <= 0;
   else if ((J == 1) && (K == 0)) Q <= 1;
   else                           Q <= ~Q;
end

endmodule
```