EEE 2243
Digital System Design
Semester II 2011/12

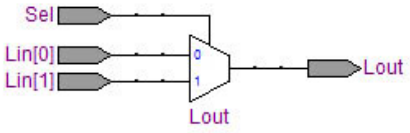
Tutorial 4

Modular Verilog

1. a. Write a module of 2-to-1 multiplexer in Verilog.

```
module Mux2(Lin, Sel, Lout);

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

assign Lout = (Sel) ? Lin[1] : Lin[0];

endmodule
```

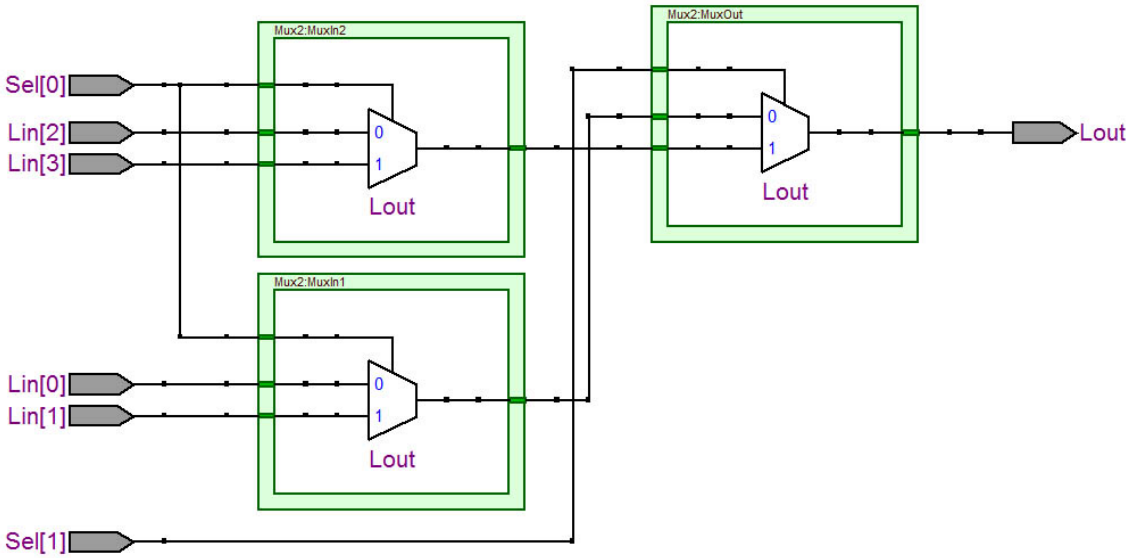
RTL View:



b. Reuse 3 of the module in (a) to construct a 4-to-1 multiplexer (with or without new added components).

```
module Mux4(Lin, Sel, Lout);

input [3:0] Lin;
input [1:0] Sel;
output Lout;
wire fromMuxIn1, fromMuxIn2;

Mux2 MuxIn1(.Lin({Lin[1], Lin[0]}),
            .Sel(Sel[0]),
            .Lout(fromMuxIn1));
Mux2 MuxIn2(.Lin({Lin[3], Lin[2]}),
            .Sel(Sel[0]),
            .Lout(fromMuxIn2));
Mux2 MuxOut(.Lin({fromMuxIn2, fromMuxIn1}),
            .Sel(Sel[1]),
            .Lout(Lout));

endmodule
```

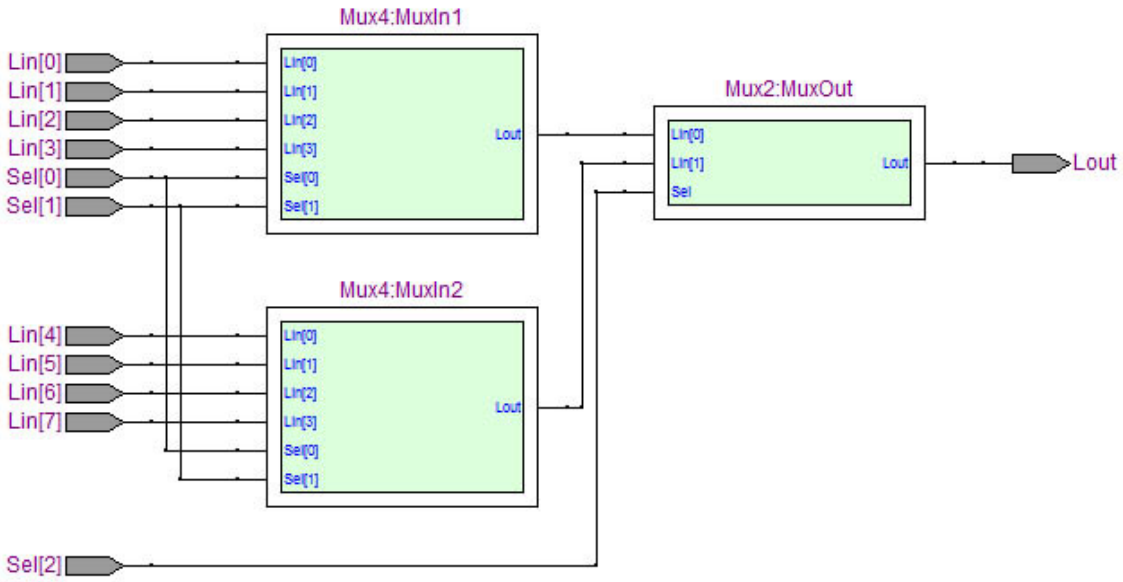
RTL View:



c. Using 2 modules of (b) and 1 module of (a), construct an 8-to-1 multiplexer (with or without new added components).

```
module Mux8(Lin, Sel, Lout);

input [7:0] Lin;
input [2:0] Sel;
output Lout;
wire fromMuxIn1, fromMuxIn2;

Mux4 MuxIn1(.Lin({Lin[3], Lin[2], Lin[1], Lin[0]}),
            .Sel({Sel[1], Sel[0]}),
            .Lout(fromMuxIn1));
Mux4 MuxIn2(.Lin({Lin[7], Lin[6], Lin[5], Lin[4]}),
            .Sel({Sel[1], Sel[0]}),
            .Lout(fromMuxIn2));
Mux2 MuxOut(.Lin({fromMuxIn2, fromMuxIn1}),
            .Sel(Sel[2]),
            .Lout(Lout));

endmodule
```

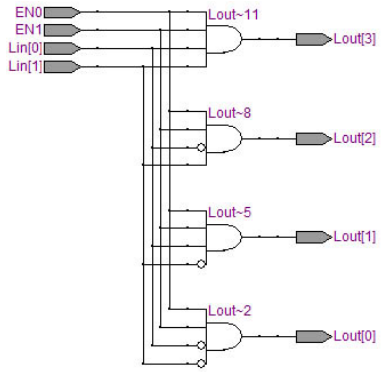
RTL View:



2. a. Write a module of 2-to-4 active high decoder in Verilog with two active high control inputs called EN0 and EN1 that passes the decoder normal output if both of them are 1. Otherwise, the decoder's outputs are fixed to all 0-s (not high-z).

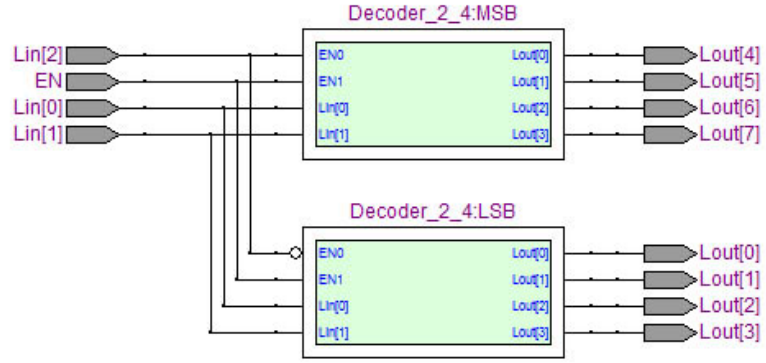
```
module Decoder_2_4(Lin, Lout, EN0, EN1);

input EN1, EN0;
input [1:0] Lin;
output [3:0] Lout;

assign Lout[0] = ~Lin[1] & ~Lin[0] & EN1 & EN0;
assign Lout[1] = ~Lin[1] &  Lin[0] & EN1 & EN0;
assign Lout[2] =  Lin[1] & ~Lin[0] & EN1 & EN0;
assign Lout[3] =  Lin[1] &  Lin[0] & EN1 & EN0;

endmodule
```

RTL View:



b. Reuse 2 of the module in (a) to construct a 3-to-8 decoder (with or without new added components).

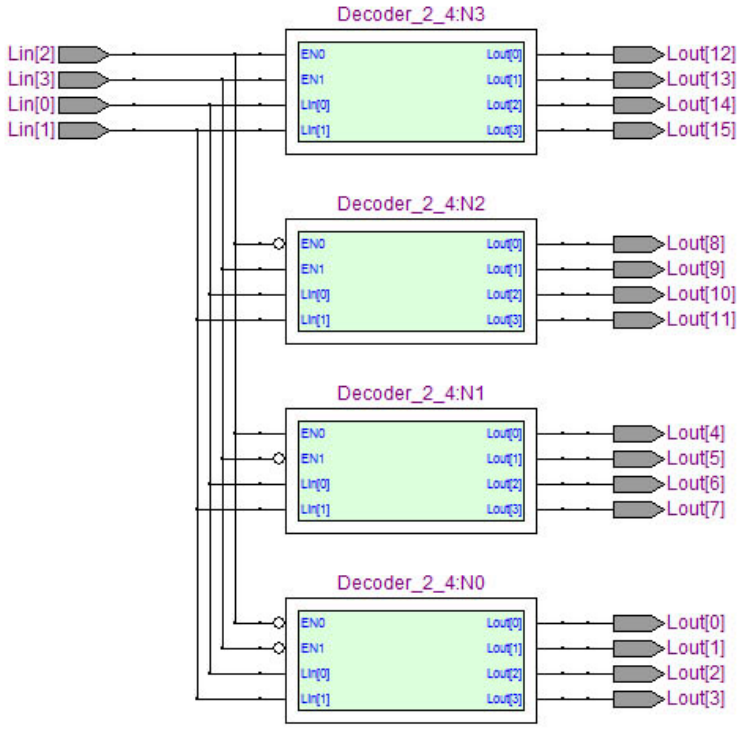
```
module Decoder_3_8(Lin, Lout, EN);

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

Decoder_2_4 LSB(.Lin({Lin[1], Lin[0]}),
                .Lout({Lout[3], Lout[2], Lout[1], Lout[0]}),
                .EN0(~Lin[2]),
                .EN1(EN));
Decoder_2_4 MSB(.Lin({Lin[1], Lin[0]}),
                .Lout({Lout[7], Lout[6], Lout[5], Lout[4]}),
                .EN0(Lin[2]),
                .EN1(EN));

endmodule
```



c. Using 4 modules in (a) to construct a 4-to-16 decoder (with or without new added components).

```
module Decoder_4_16(Lin, Lout);

input [3:0] Lin;
output [15:0] Lout;

Decoder_2_4 N0(.Lin({Lin[1], Lin[0]}),
               .Lout({Lout[3], Lout[2], Lout[1], Lout[0]}),
               .EN0(~Lin[2]),
               .EN1(~Lin[3]));
Decoder_2_4 N1(.Lin({Lin[1], Lin[0]}),
               .Lout({Lout[7], Lout[6], Lout[5], Lout[4]}),
               .EN0( Lin[2]),
               .EN1(~Lin[3]));
Decoder_2_4 N2(.Lin({Lin[1], Lin[0]}),
               .Lout({Lout[11], Lout[10], Lout[9], Lout[8]}),
               .EN0(~Lin[2]),
               .EN1( Lin[3]));
Decoder_2_4 N3(.Lin({Lin[1], Lin[0]}),
               .Lout({Lout[15], Lout[14], Lout[13], Lout[12]}),
               .EN0( Lin[2]),
               .EN1( Lin[3]));

endmodule
```

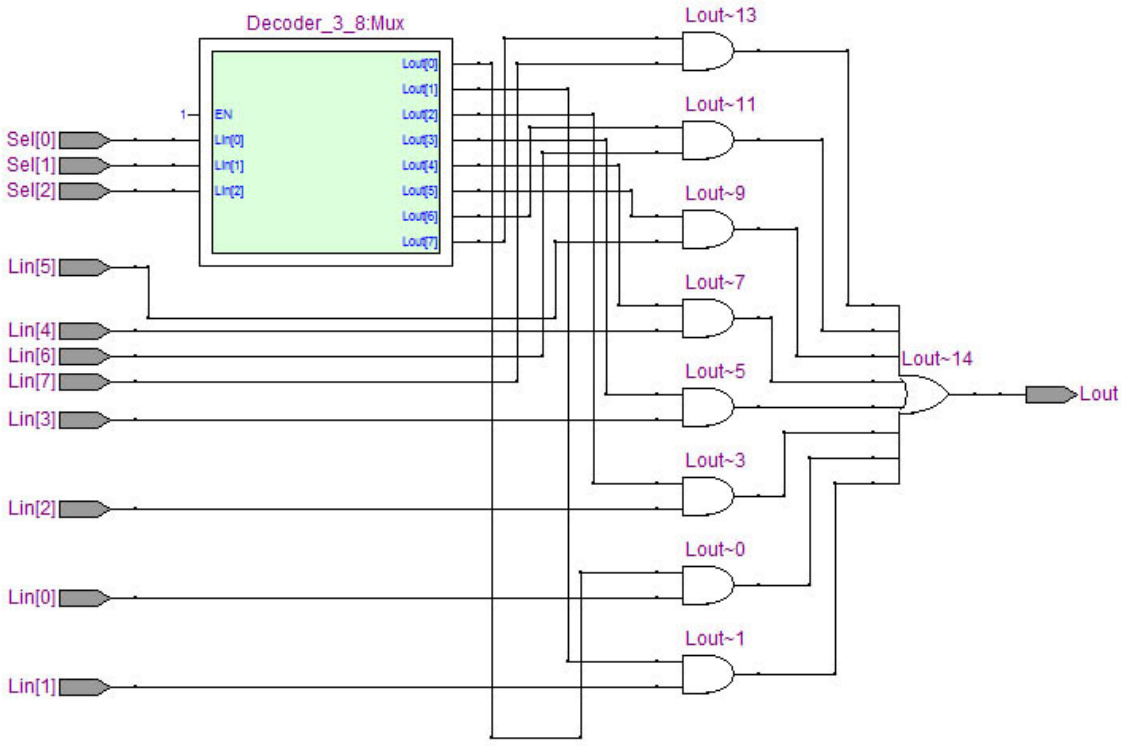
RTL View:



3. Use the decoder in (2.b) to construct an 8-to-1 multiplexer (with or without new added components).

```
module Mux8(Lin, Sel, Lout);

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

Decoder_3_8 Mux(.Lin(Sel), .Lout(w), .EN(1));

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

endmodule
```

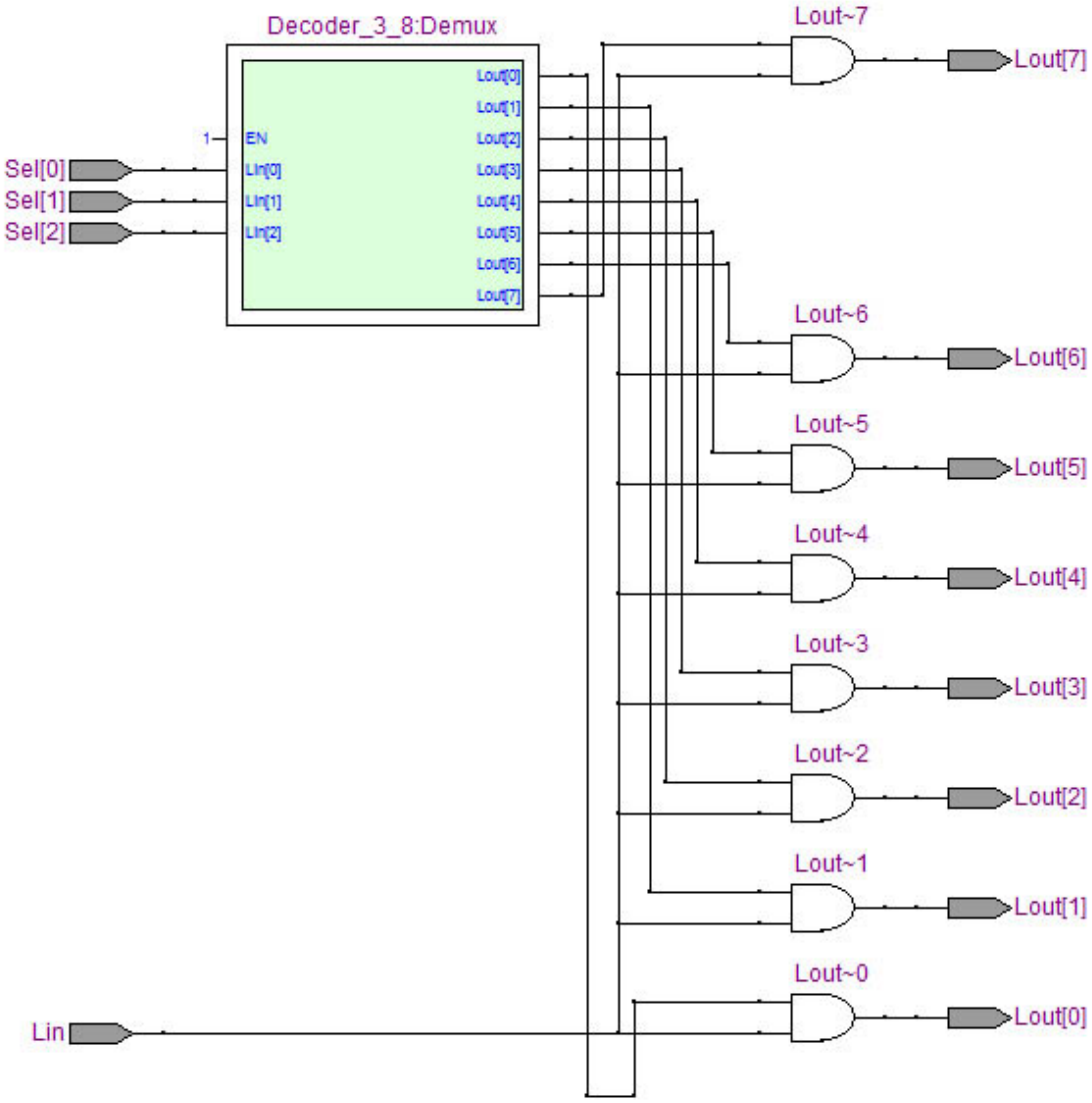
RTL View:



4. Use the decoder in (2.b) to construct a 1-to-8 demultiplexer (with or without new added components).

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

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

Decoder_3_8 Demux(.Lin(Sel), .Lout(w), .EN(1));

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

endmodule
```

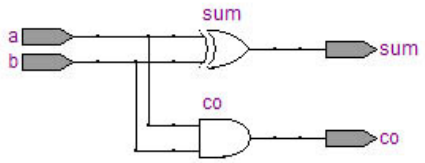
RTL View:



5. As shown in the class, write a module of 1-bit full-adder from two modules of half-adders. Then from the 1-bit full-adder module, construct a 4-bit full-adder using ripple carry method.

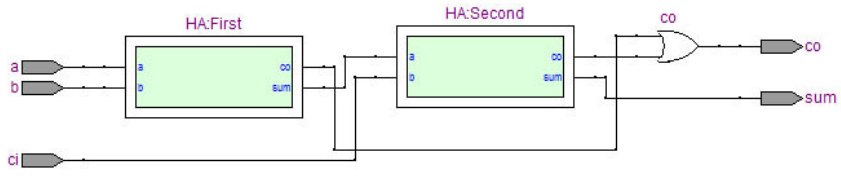
```
module HA(a, b, sum, co);

input a, b;
output sum, co;

assign sum = a ^ b;
assign co = a & b;

endmodule
```

RTL View:




```
module FA(a, b, ci, sum, co);

input a, b, ci;
output sum, co;
wire w1, w2, w3;

HA First(.a(a), .b(a), .sum(w1), .co(w2));
HA Second(.a(w1), .b(ci), .sum(sum), .co(w3));

assign co = w2 | w3;

endmodule
```

RTL View:



```
module FA4(a, b, ci, sum, co);

input ci;
input [3:0] a;
input [3:0] b;
output [3:0] sum;
output co;

FA FA0(.a(a[0]), .b(b[0]), .ci(ci), .sum(sum[0]), .co(w1));
FA FA1(.a(a[1]), .b(b[1]), .ci(w1), .sum(sum[1]), .co(w2));
FA FA2(.a(a[2]), .b(b[2]), .ci(w2), .sum(sum[2]), .co(w3));
FA FA3(.a(a[3]), .b(b[3]), .ci(w3), .sum(sum[3]), .co(co));

endmodule
```

RTL View: