1) Verilog code with comments for the 2:4 binary decoder, the 4:2 binary encoder, and the 4:2 priority encoder. Do not use behavioral Verilog for these descriptions! Use the structural and dataflow concepts introduced in the previous lab.

2:4 binary decoder :

```
module two_four_decoder(
   input  wire [1:0] W,  // 2-bit input
   input  wire En,       // Enable signal
   output wire [3:0] Y   // 4-bit one-hot output
);

   // Each output is defined by its Boolean expression
   assign Y[0] = En & ~W[1] & ~W[0];  // When W = 00
   assign Y[1] = En & ~W[1] &  W[0];   // When W = 01
   assign Y[2] = En &  W[1] & ~W[0];   // When W = 10
   assign Y[3] = En &  W[1] &  W[0];    // When W = 11
endmodule
```

Table 1: 2:4 Binary Decoder Truth Table

En	w_1	w_0	y_3	y_2	y_1	y_0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0
0	X	X	0	0	0	0

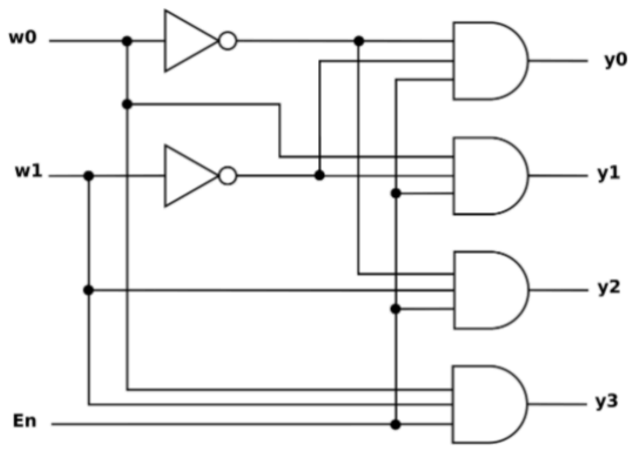


Figure 1: 2:4 Binary Decoder Gate-level Schematic

4:2 binary encoder:

```
module four_two_encoder(
    input  wire [3:0] W,   // 4-bit input
    output wire       zero, // 1 if all inputs are 0, else 0
    output wire [1:0] Y    // 2-bit output
);
    // Intermediate signals
    wire or_32;        // W[3] OR W[2]
    wire or_31;        // W[3] OR W[1]
    wire any_high;     // W[3] OR W[2] OR W[1] OR W[0]

    // zero = NOT( W[3] OR W[2] OR W[1] OR W[0] )
    or  (any_high, W[3], W[2], W[1], W[0]);
    not (zero, any_high);

    // Y[1] = W[3] OR W[2]
    or  (or_32, W[3], W[2]);
    assign Y[1] = or_32;

    // Y[0] = W[3] OR W[1]
    or  (or_31, W[3], W[1]);
    assign Y[0] = or_31;

endmodule
```

Table 2: 4:2 Binary Encoder Truth Table

w_3	w_2	w_1	w_0	y_1	y_0	$zero$
0	0	0	0	X	X	1
0	0	0	1	0	0	0
0	0	1	0	0	1	0
0	1	0	0	1	0	0
1	0	0	0	1	1	0

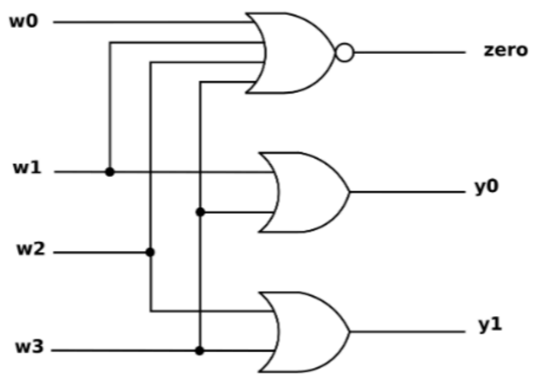
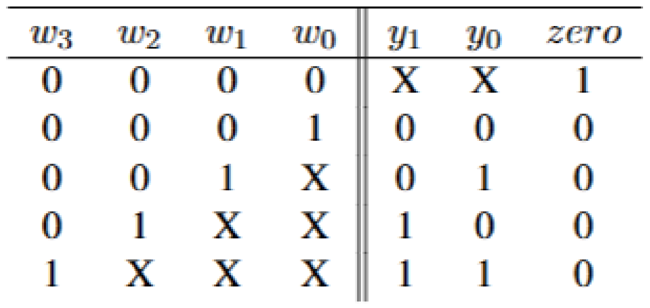


Figure 2: 4:2 Binary Encoder Gate-level Schematic

4:2 priority encoder.

```
module priority_encoder(
    input  wire [3:0] W,   // 4-bit input
    output wire       zero, // High if all inputs are 0
    output wire [1:0] Y    // 2-bit encoded output
);

    // 'zero' is asserted when all bits are zero.
    assign zero = ~(W[3] | W[2] | W[1] | W[0]);

    // Y[1] is 1 if W[3] or W[2] is high.
    assign Y[1] = W[3] | W[2];

    // Y[0] is 1 if either W[3] is high or (W[3] and W[2] are low and W[1] is high).
    assign Y[0] = W[3] | ((~W[3] & ~W[2]) & W[1]);

endmodule
```

Table 3: 4:2 Priority Encoder Truth Table



w_3	w_2	w_1	w_0	y_1	y_0	$zero$
0	0	0	0	X	X	1
0	0	0	1	0	0	0
0	0	1	X	0	1	0
0	1	X	X	1	0	0
1	X	X	X	1	1	0

2) The complete truth table for the gate-level schematic shown in Figure 2. This truth table should not include “don’t cares” (i.e. X)!

W3	W2	W1	W0	Y1	Y0	Zero
0	0	0	0	0	0	1
0	0	0	1	0	0	0
0	0	1	0	0	1	0
0	1	0	0	1	0	0
1	0	0	0	1	1	0

Zero is 1 only when all inputs are 0

3)A brief comparison of the behavioral implementation of a multiplexer described in the background section with the multiplexer you described in the previous lab using structural and dataflow

Designing a multiplexer using structural Verilog provides clear gate representation, which aids in visualizing the hardware. In contrast, dataflow design focuses on functionality, with no insight into the gate. While structural implementation is preferred for understanding the physical design, it lacks the synthesis flexibility that dataflow offers. However, the lack of specific gate insights in data flow design makes it more appropriate for brief designs.

Multiplexer using structural	**Multiplexer using data flow design**
The design uses the multiplexer using basic logic gate	The design directly assigning the functionality towards the output
It overall helps the synthesis tool to implement the hardware part with the mentioned gates	The synthesis tool is flexible enough to work with any gates and can recognize that you are attempting to implement the mux from the code