Ecen 248 Pre-Lab

Joaquin Salas

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.

4:2 Priority Encoder

```
//declare the inputs and outputs for the two to for decoder
module two_four_decoder(
  input wire [1:0] W,
  input wire En,
  output wire [3:0] Y
);

  wire [1:0] nW, nEn; //implement the not gates on the circuit
  assign nEn = ~En; //for Enable assign nEn
  assign nW[0] = ~nW[0]; //for nW0 assign the function neccessary
  assign nW[1] = ~nW[1]; //Do the same for nW1 and assign a NOT gate

  //assign output values
  //After finding a schematic of the two to four decoder
  assign Y[0] = ~(nEn & nW[0] & nW[1]); //nEn & nW0 & nW1
  assign Y[1] = ~(nEn & nW[0] & W[1]); //nEn & nW0 & nW1
  assign Y[2] = ~(nEn & W[0] & nW[1]); //nEn & nW0 & nW1
  assign Y[3] = ~(nEn & W[0] & W[1]); //nEn & nW0 & nW1

endmodule
```

4:2 encoder

```
module four_two_encoder(
  input wire [3:0] W,
  output wire [1:0] Y,
  output wire zero
);


  nor nor1(zero, W[0], W[1], W[2], W[3]);
  or or0(Y[0], W[1], W[3]);
  or or1(Y[1], W[3], W[2]);

endmodule
```

Priority encoder

```
module priority_encoder(
  input wire [3:0] W,
  output wire [1:0] Y,
  output wire zero
);
  wire W2bar;
  wire and_out;

  not not0(W2bar, W[2]);
  and and0(and_out, W2bar, W[1]);
  or or0(Y[1], W[3], W[2]);
  or or1(Y[0], and_out, W[3]);
  or or2(Y[0], W[1], W[3]);
  or or3(Y[1], W[3], W[2]);
  nor nor1(zero, W[0], W[1], W[2], W[3]);

endmodule
```

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') as outputs!

$y_0 = w_1 + w_3$ $y_1 = w_2 + w_3$

W_3	W_2	W_1	W_0	Y_0	Y_1
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	1	0
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	1	1
0	1	1	1	1	1
1	0	0	0	1	1
1	0	0	1	1	1
1	0	1	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1
1	1	0	1	1	1
1	1	1	0	1	1
1	1	1	1	1	1

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.

Structural design multiplexers are a great way to visualize what hardware, layout, and gate are used in physical circuit. Dataflow code design multiplexer are a good way to understand how the data in the circuit from looking at the outputs and seeing what gates and commands are used. Behavioral code design multiplexer is to let the designer to understand the circuit completely.