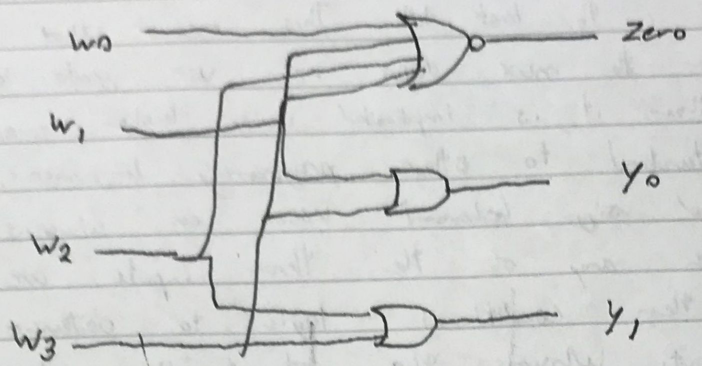
Drew DeHaven
Section 508

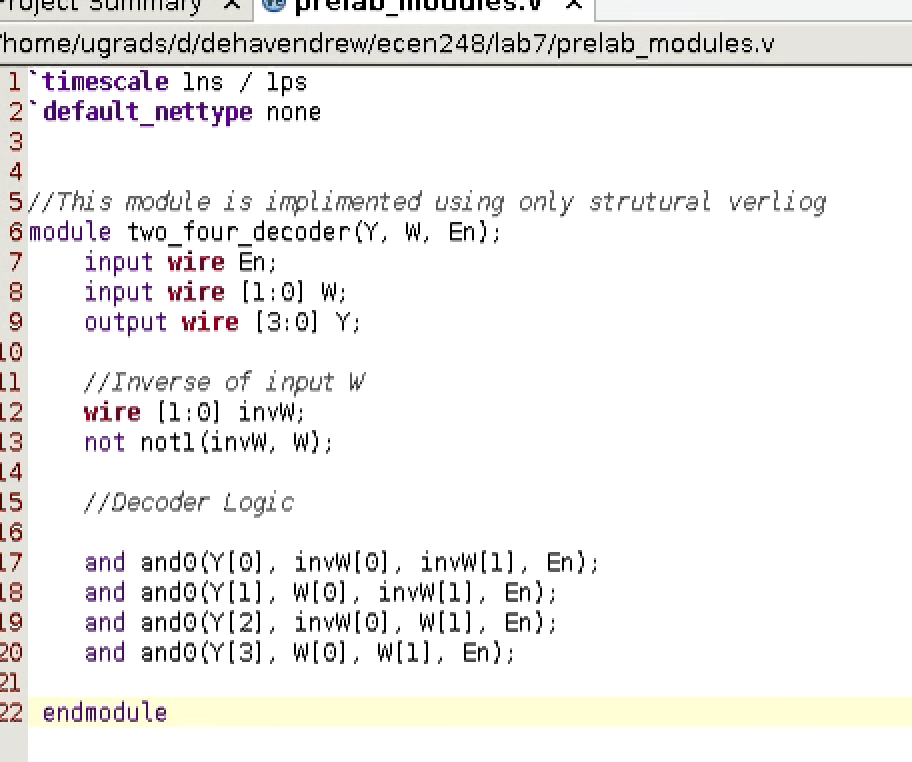
Prelab 7

2. Truth table for



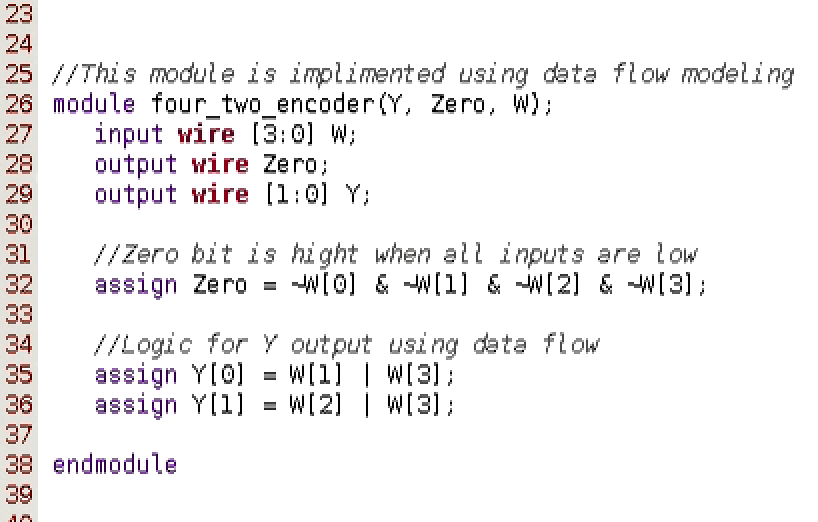
W_3	W_2	W_1	W_0	Y_1	Y_0	Zero
0	0	0	0	0	0	1
0	0	0	1	0	0	0
0	0	1	0	0	1	0
0	0	1	1	0	1	0
0	1	0	0	1	0	0
0	1	0	1	1	0	0
0	1	1	0	1	1	0
0	1	1	1	1	1	0
1	0	0	0	1	1	0
1	0	0	1	1	1	0
1	0	1	0	1	1	0
1	0	1	1	1	1	0
1	1	0	0	1	1	0
1	1	0	1	1	1	0
1	1	1	0	1	1	0
1	1	1	1	1	1	0

3 The two-to-one mux in this lab is written using behavioral verilog, rather that structural verilog as was done in the last lab. This means that the new design for the mux does not use gate level implementation, rather it is implemented using higher level features, more standard to other programming languages. The mux designed using behavioral uses an always block to detect when any of the three inputs are changed, and then conditional logic to determine the correct output. Whereas the structurally designed mux uses logic gates to construct every possible output. Thus it is apparent that behavioral verilog is far more scalable. Additionaly, the ~~[illegible]~~ structural mux has a wire output, while the behavioral uses a register, which makes it suited for sequential logic.

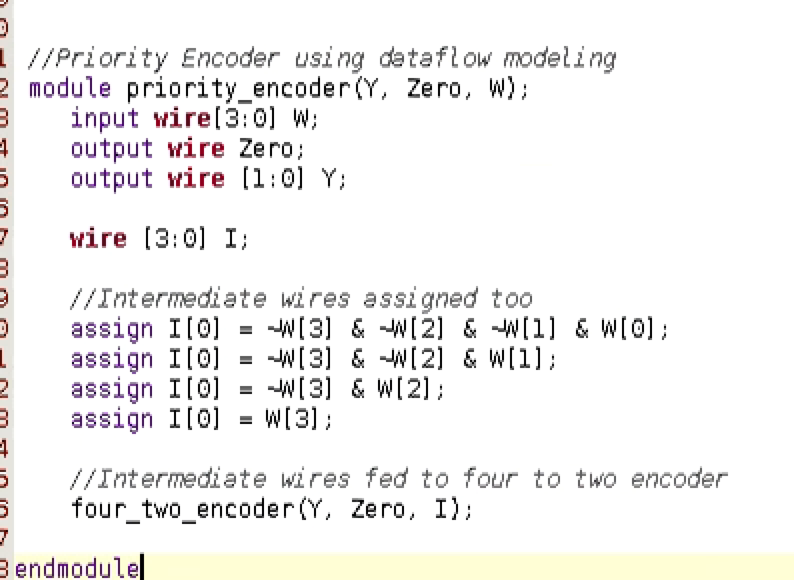


/home/ugrads/d/dehavendrew/ecen248/lab7/prelab_modules.v

```
`timescale 1ns / 1ps
`default_nettype none


//This module is implimented using only strutural verliog
module two_four_decoder(Y, W, En);
    input wire En;
    input wire [1:0] W;
    output wire [3:0] Y;

    //Inverse of input W
    wire [1:0] invW;
    not not1(invW, W);

    //Decoder Logic

    and and0(Y[0], invW[0], invW[1], En);
    and and0(Y[1], W[0], invW[1], En);
    and and0(Y[2], invW[0], W[1], En);
    and and0(Y[3], W[0], W[1], En);

 endmodule
```



```verilog


//This module is implimented using data flow modeling
module four_two_encoder(Y, Zero, W);
   input wire [3:0] W;
   output wire Zero;
   output wire [1:0] Y;

   //Zero bit is hight when all inputs are low
   assign Zero = ~W[0] & ~W[1] & ~W[2] & ~W[3];

   //Logic for Y output using data flow
   assign Y[0] = W[1] | W[3];
   assign Y[1] = W[2] | W[3];

endmodule

```



```
//Priority Encoder using dataflow modeling
module priority_encoder(Y, Zero, W);
    input wire[3:0] W;
    output wire Zero;
    output wire [1:0] Y;

    wire [3:0] I;

    //Intermediate wires assigned too
    assign I[0] = ~W[3] & ~W[2] & ~W[1] & W[0];
    assign I[0] = ~W[3] & ~W[2] & W[1];
    assign I[0] = ~W[3] & W[2];
    assign I[0] = W[3];

    //Intermediate wires fed to four to two encoder
    four_two_encoder(Y, Zero, I);

endmodule
```