Justin VanWinkle
EEE 3342C
Experiment #6
19 July 2015

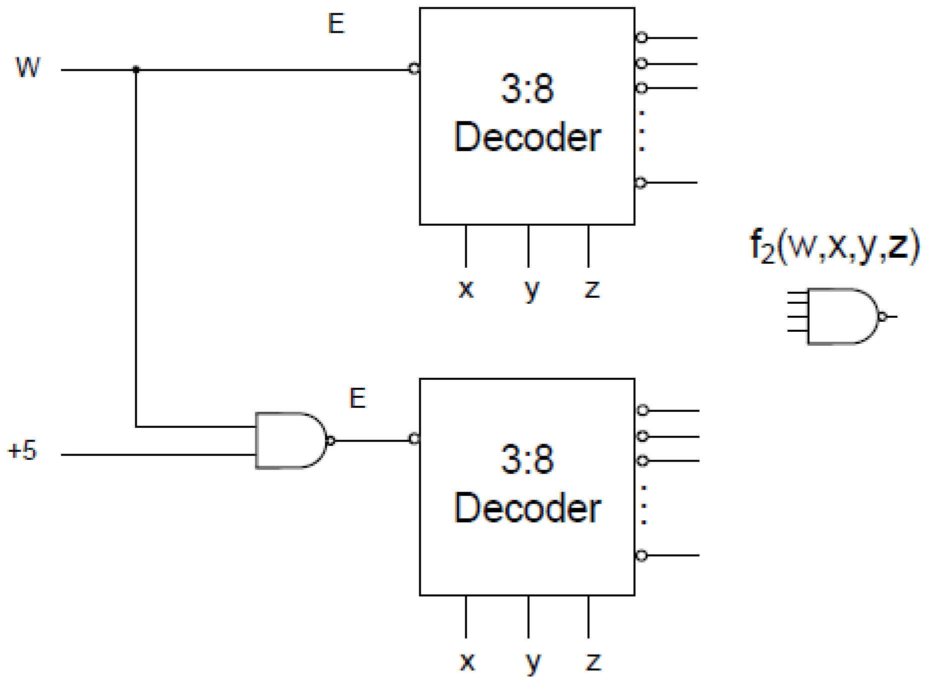
Justin VanWinkle
19 June 2015
Experiment #6

Objective

This experiment is intended to introduce decoders and their use in selecting one output at a time using the Xilinx ISE design tools. The logic gate will be analyzed using the Xilinx ISE simulation tool as well as on a BASYS 2 FPGA development board.

Diagrams

The following is a block diagram that shows a block diagram of two 3 to 8 decoders being used to implement a 4 to 16 decoder which exemplifies the usage of decoders in this experiment:



As can be seen, this decoder uses 3 bits for an output selection while a fourth bit is used as an enabler to select between the two decoders.

Equipment

The equipment used for this experiment included a computer with the Xilinx ISE design suite and Digilent ExPort installed as well as a Digilent BAYSYS 2 development board.

Procedure

Part 1

The first step in this experiment should be analyzing and understanding what our expected output should be based on any given set of inputs. This will give us a better understanding of how to build the schematic of the circuit. The below table is a comprehensive view of the logic possibilities in the circuit at any given moment, matched with their respective output

Justin VanWinkle
19 June 2015
Experiment #6

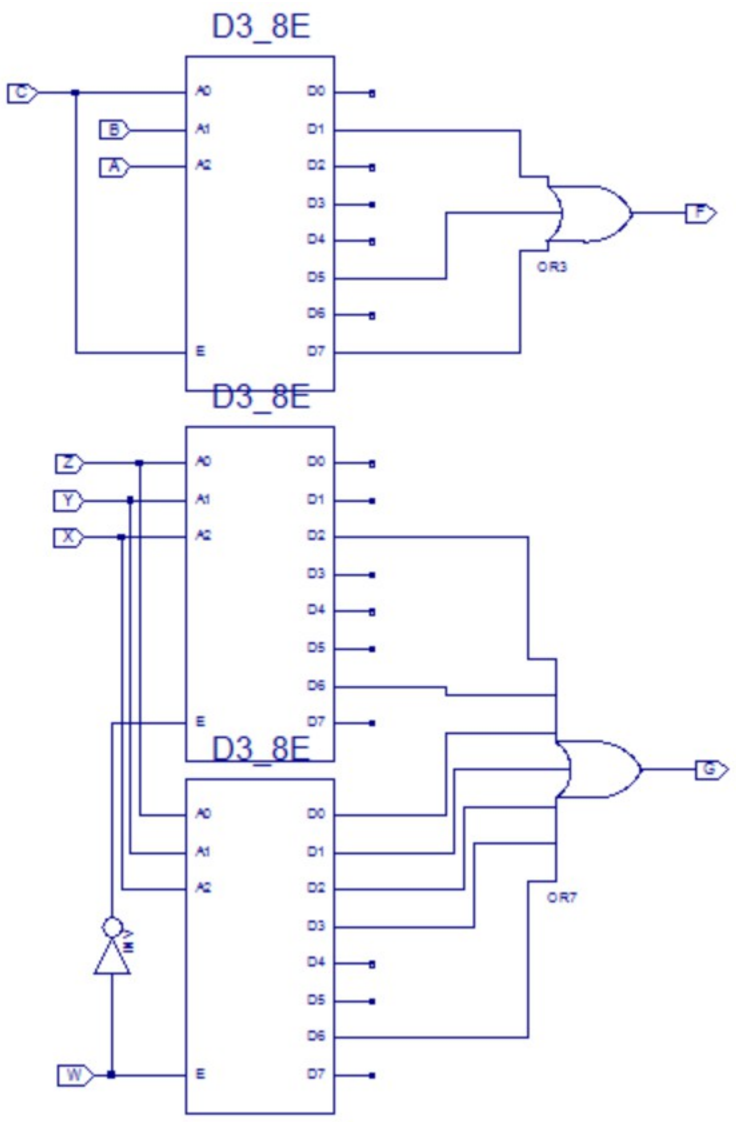
f(a,b,c) = (a+b')c	a	b	c	f
	0	0	0	0
	0	0	1	1
f=m1,m5,m7	0	1	0	0
	0	1	1	0
	1	0	0	0
	1	0	1	1
	1	1	0	0
	1	1	1	1

g(w,x,y,z) = wx'+yz'
g=m2,m6,m8,m9,m10,m11,m14

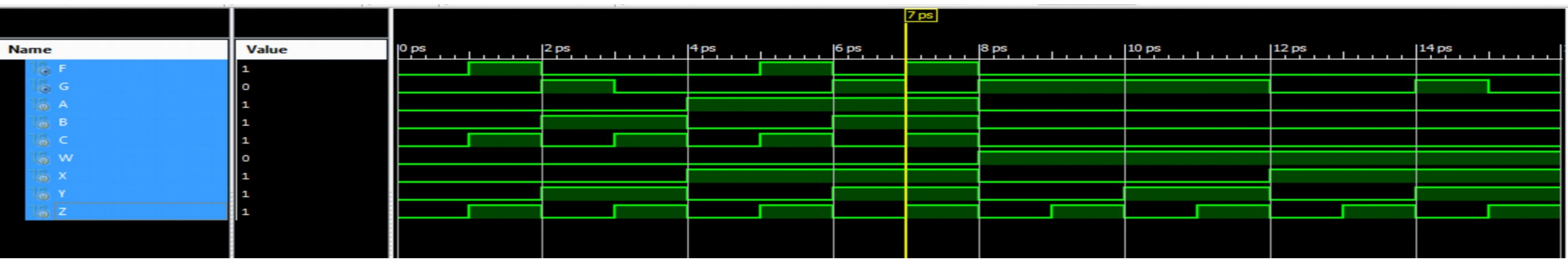
w	x	y	z	g
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

The above logic diagram shows the expected output of the decoders of this experiment based on all possible combinations of inputs.

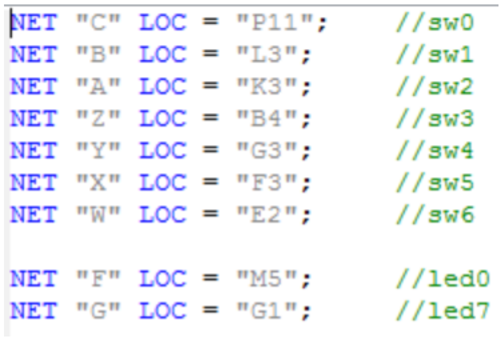
Next, the following schematic was created using the data from the logic table:



After completion of the above schematic capture, the operator then synthesized and implemented the design. Using the isim interface, the operator next simulated and tested the schematic design for its accuracy based on all possible combinations of input and function selects. The result of this simulation is as follows:



With a proper simulation result, the operator next created constraints that mapped each input and output to specific pins of the CP132. The constraints were:



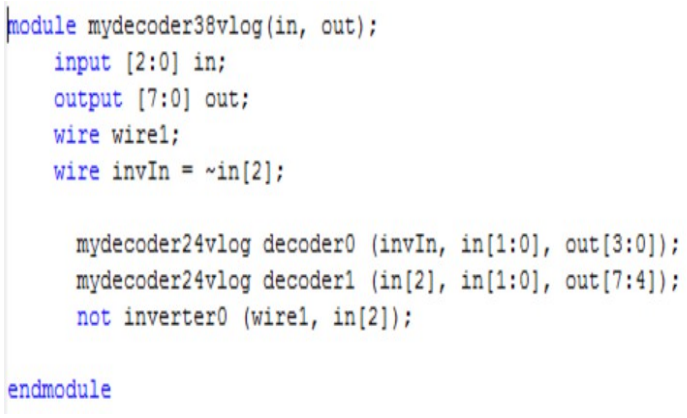
```
NET "C" LOC = "P11";      //sw0
NET "B" LOC = "L3";       //sw1
NET "A" LOC = "K3";       //sw2
NET "Z" LOC = "B4";       //sw3
NET "Y" LOC = "G3";       //sw4
NET "X" LOC = "F3";       //sw5
NET "W" LOC = "E2";       //sw6

NET "F" LOC = "M5";       //led0
NET "G" LOC = "G1";       //led7
```

Now, the operator was ready to implement the design a final time and create the .bit file used to program the BASYS 2 board. To transfer the file to the BASYS 2 board, Digilent Adept was used. Once in Adept, with the BASYS 2 board powered on, the operator selected the initialize chain button to establish communication between the computer and the BASYS 2 board. Next, the operator used the browse button next to the “FPGA” text to select the newly created *.bit file for programming the BASYS 2 board. Finally, the “program chain” button was used to push the program to the BASYS 2 board and the operator could test the program on the board for proper output.

Part 2

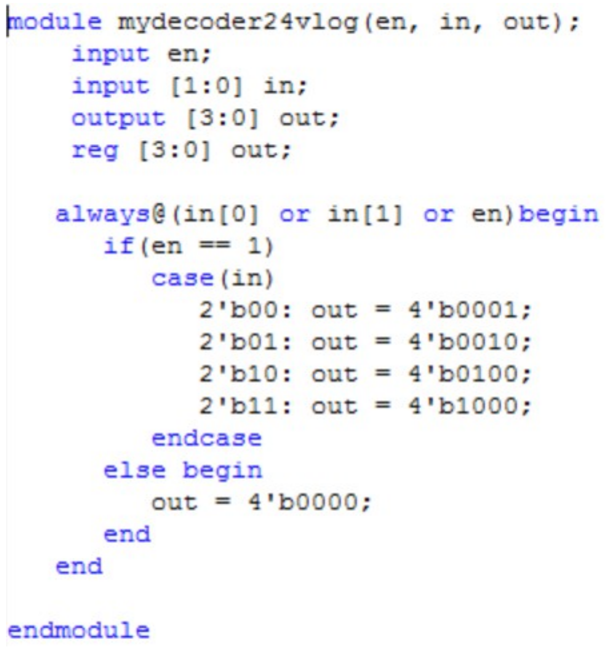
For the second part of the experiment, the operator used verilog to implement a 3 to 8 decoder using two 2 to 4 decoders. This very slightly changed the process up to the point of creating a simulation, but beyond that, the process is identical.

When creating a VERILOG module, the user will select inputs and outputs based on the need of the problem and will also select a naming convention for each. The code for the 3 to 8 decoder can be seen to the right:



```
module mydecoder38vlog(in, out);
    input [2:0] in;
    output [7:0] out;
    wire wire1;
    wire invIn = ~in[2];

      mydecoder24vlog decoder0 (invIn, in[1:0], out[3:0]);
      mydecoder24vlog decoder1 (in[2], in[1:0], out[7:4]);
      not inverter0 (wire1, in[2]);

endmodule
```

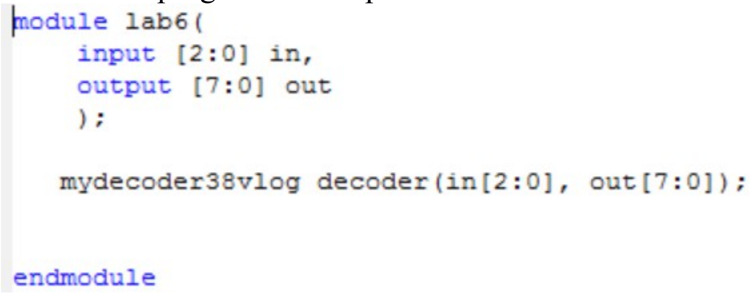
Justin VanWinkle
19 June 2015
Experiment #6



```
module mydecoder24vlog(en, in, out);
    input en;
    input [1:0] in;
    output [3:0] out;
    reg [3:0] out;

  always@(in[0] or in[1] or en)begin
     if(en == 1)
        case(in)
           2'b00: out = 4'b0001;
           2'b01: out = 4'b0010;
           2'b10: out = 4'b0100;
           2'b11: out = 4'b1000;
        endcase
     else begin
        out = 4'b0000;
     end
  end

endmodule
```

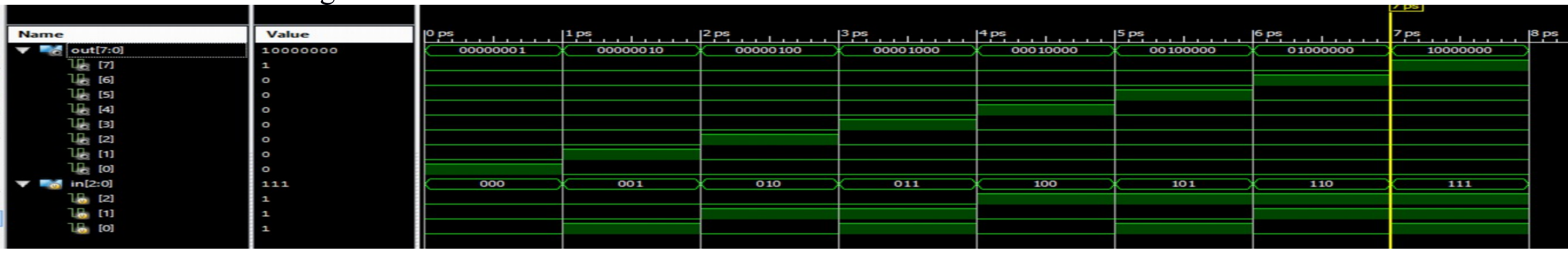
This, of course instantiates two 2 to 4 decoders whose code is at left.

A main entry point into the program was implemented like this:

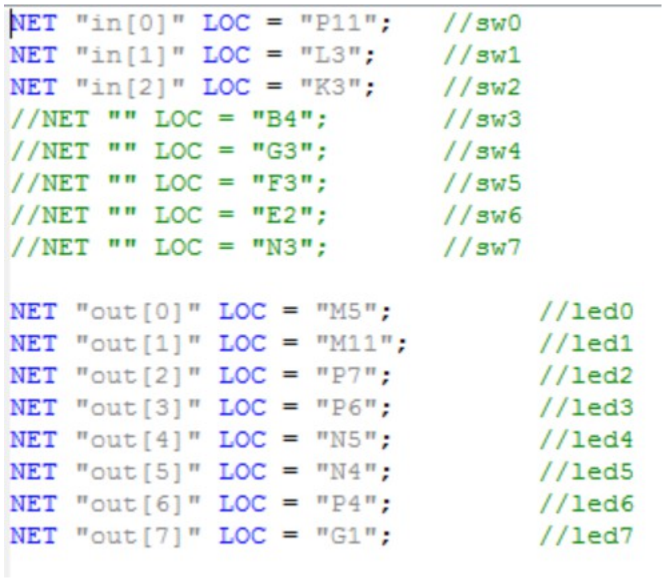


```
module lab6(
    input [2:0] in,
    output [7:0] out
    );

   mydecoder38vlog decoder(in[2:0], out[7:0]);


endmodule
```

After completion of each of the above verilog code, the operator then synthesized and implemented the design. Next, the operator used the newly written code to simulate the intended circuit for its accuracy based on all possible combinations of inputs. The resulting simulation is as follows:



With a proper simulation result, the operator next created constraints that mapped each input and output to specific pins of the CP132. The constraints were:



```
NET "in[0]" LOC = "P11";    //sw0
NET "in[1]" LOC = "L3";     //sw1
NET "in[2]" LOC = "K3";     //sw2
//NET "" LOC = "B4";        //sw3
//NET "" LOC = "G3";        //sw4
//NET "" LOC = "F3";        //sw5
//NET "" LOC = "E2";        //sw6
//NET "" LOC = "N3";        //sw7

NET "out[0]" LOC = "M5";          //led0
NET "out[1]" LOC = "M11";         //led1
NET "out[2]" LOC = "P7";          //led2
NET "out[3]" LOC = "P6";          //led3
NET "out[4]" LOC = "N5";          //led4
NET "out[5]" LOC = "N4";          //led5
NET "out[6]" LOC = "P4";          //led6
NET "out[7]" LOC = "G1";          //led7
```

Now, the operator was ready to implement the design a final time and create the .bit file used to program the BASYS 2 board. To transfer the file to the BASYS 2 board, Digilent Adept was used. Once in Adept, with the BASYS 2 board powered on, the operator selected the initialize chain button to establish communication between the computer and the BASYS 2 board. Next, the operator used the browse button next to the “FPGA” text to select the newly created *.bit file for programming the BASYS 2 baord. Finally, the “program chain” button was used to push the program to the BASYS 2 board and the operator could test the program on the board for proper output.

Expected Results of Circuit

The expected results of this circuit can be seen in the logic table previously created. To further verify proper output, the operator made assertions that checked for errors during each simulation. The code used for simulation of part 1 is as follows:

```
`timescale 1ps / 1ps

module schematic_schematic_sch_tb();

// Inputs
  reg A;
  reg B;
  reg C;
     reg W;
     reg X;
     reg Y;
     reg Z;

// Output
```

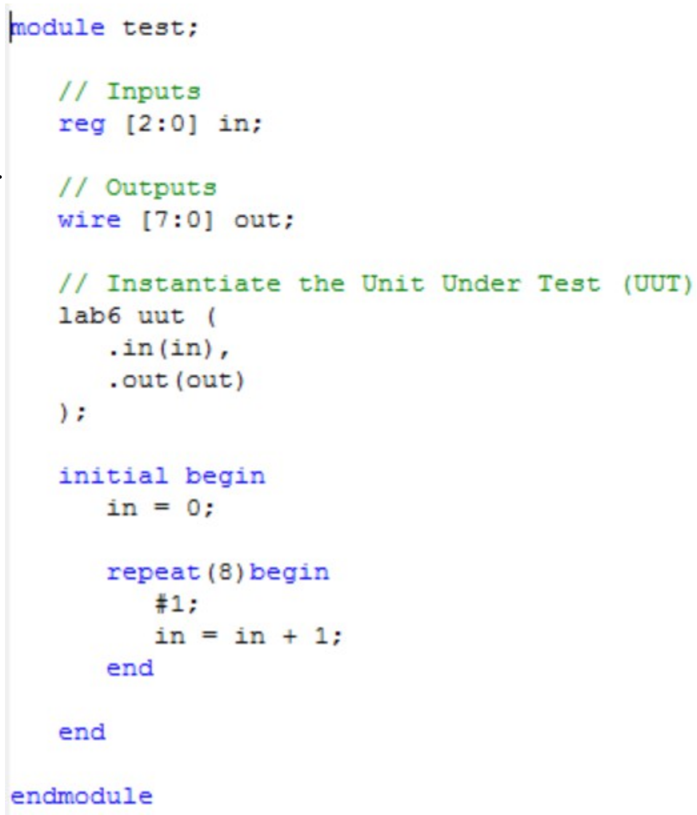
```
  wire F;
     wire G;

// Bidirs

// Instantiate the UUT
  schematic UUT (
          .A(A),
          .B(B),
          .C(C),
          .F(F),
          .W(W),
          .X(X),
          .Y(Y),
          .Z(Z),
          .G(G)
  );

  initial begin
          A = 1;
          B = 1;
          C = 1;
          W = 0;
          X = 1;
          Y = 1;
          Z = 1;
     end

     always begin
          repeat(2)begin
               A=~A;
               X=~X;
               repeat(2)begin
                    B=~B;
                    Y=~Y;
                    repeat(2)begin
                         C=~C;
                         Z=~Z;
                         #1;
                         if((A|~B)&C == 1 && F != 1)begin
                              $display("Incorrect output for F = 1", $time-1);
                         end
                         if((A|~B)&C == 0 && F != 0)begin
                              $display("Incorrect output for F = 0", $time-1);
                         end
                         if((W&~X | Y&~Z) == 1 && G != 1) begin
```

```
                        $display("Incorrect output for G = 1", $time-1);
                    end
                    if((W&~X | Y&~Z) == 0 && G != 0) begin
                        $display("Incorrect output for G = 0", $time-1);
                    end
                end
            end
        end
        A = 0;
        B = 0;
        C = 0;
        W = 1;
        repeat(2)begin
            X=~X;
            repeat(2)begin
                Y=~Y;
                repeat(2)begin
                    Z=~Z;
                    #1;
                    if((W&~X | Y&~Z) == 1 && G != 1) begin
                        $display("Incorrect output for G = 1", $time-1);
                    end
                    if((W&~X | Y&~Z) == 0 && G != 0) begin
                        $display("Incorrect output for G = 0", $time-1);
                    end

                end
            end
        end
    end
endmodule
```

As can be seen, during the simulation, the developer created assumptions that checked for proper response based on the logic chart at the beginning of this report. A message was shown to report any unexpected outputs. This serves as a fail-safe way of testing.

The test code for part 2 is to the right:

No block was written to test for proper output since the simulation output of this code was extremely simple to evaluate.



```
module test;

   // Inputs
   reg [2:0] in;

   // Outputs
   wire [7:0] out;

   // Instantiate the Unit Under Test (UUT)
   lab6 uut (
      .in(in),
      .out(out)
   );

   initial begin
      in = 0;

      repeat(8)begin
         #1;
         in = in + 1;
      end

   end

endmodule
```

Design Specification Plan

The results of this experiment draw a sound conclusion that the objective has been met by the execution of the procedure. This is given by the fact that simulations and test results match verbatim to that of the expected results based on the calculations performed beforehand. The methodology chosen for this experiment followed a calculation-first route such that the operator would know what to expect. This also provided simplicity for the operator when designing the schematic as well as the VERILOG module. Given that a successful attempt was made to design, simulate, and implement a function using a multiplexer, which was the objective of this experiment, this experiment has met all of its requirements.

Test Plan

This experiment should be tested and verified by the following means. The tester should obtain a BASYS 2 development board with this program loaded on it. Once the tester has set up the board and powered it on, he should test for output based on the provided logic table. Additionally, the tester can run a simulation at any time and watch for console output informing him of wrong outputs.

Results

Please see the section entitled procedures for simulation results.
The outcome of this experiment accurately matched that of the calculations performed and presented in the expected results section of this report. This means that a parallel adder was successfully designed and implemented, because when it was given two numbers, the parallel adder was able to produce and return the sum of the given values.

Conclusion

In conclusion, this experiment as well as the report was a huge waste of time, similar to the rest. Nevertheless, we can see from it all that decoders have a useful place in logic design for the sake of things such as demultiplexing and memory address decoding.

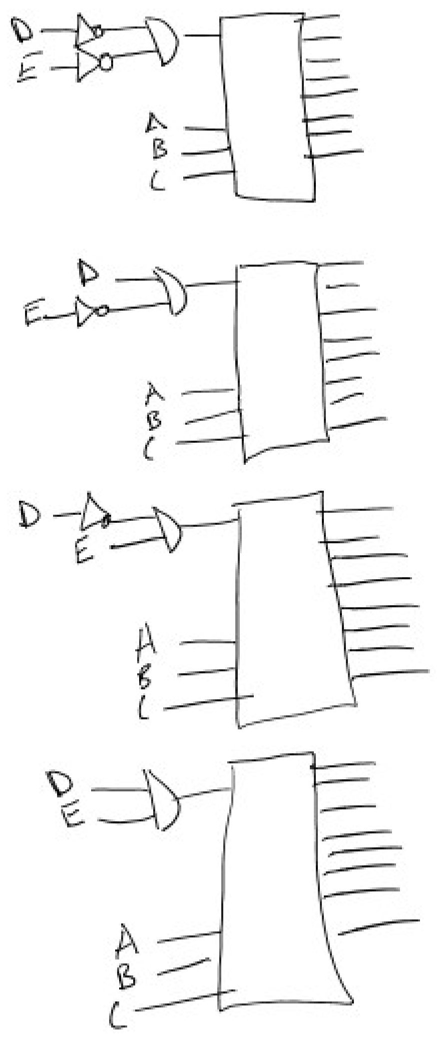
Questions

1. An OR gate with inverted inputs is the equivalent of a NAND gate without inverted inputs. The inverse of this statement is also true.

2. A multiplexer selecting the specific output function would be the best choice here. A design similar to an ALU would likely be most significant.
   Yes, I have met all requirements for this lab. See above section entitled design specification plan for more information.

3. SSI and MSI are beneficial for minimizing footprint by optimizing the number of pins. Unfortunately, this creates major limitations in making changes to the device whereas an FPGA doesn't have this limitation.

4. An AND-gate would be needed to select the desired decoder based on two bits. Rough schematic to right:



5. Test plan is above in section called “Test Plan”

6. See “Design Specification Plan” above. All products function as desired and intended. Therefore, all requirements of this lab have been completed