

Brodrick Young
ECEN 340
Lab 2 – Vivado Intro / Structural Decoder

Purpose:

1. To review the Basys 3 board
2. To review Vivado usage
3. To implement a simple structural Verilog design

Procedure:

Part 1 (First Lab Day). Download the Basys3 Vivado Decoder Tutorial. Follow the instructions in the tutorial until you have a functioning decoder.

The image shows a screenshot of the Vivado IDE interface. On the left, the Verilog code for a decoder module is displayed. The code defines an 8-bit output register 'led' based on a 3-bit input 'sw'. The code is as follows:

```
15 //  
16 // Revision:  
17 // Revision 0.01 - File Created  
18 // Additional Comments:  
19 //  
20 //////////////////////////////////////  
21  
22  
23 module decoder(  
24     input [2:0] sw,  
25     output reg [7:0] led  
26 );  
27  
28     always @ (sw)  
29     case (sw)  
30         3'b000: led=8'h01;  
31         3'b001: led=8'h02;  
32         3'b010: led=8'h04;  
33         3'b011: led=8'h08;  
34         3'b100: led=8'h10;  
35         3'b101: led=8'h20;  
36         3'b110: led=8'h40;  
37         3'b111: led=8'h80;  
38     endcase  
39 endmodule  
40
```

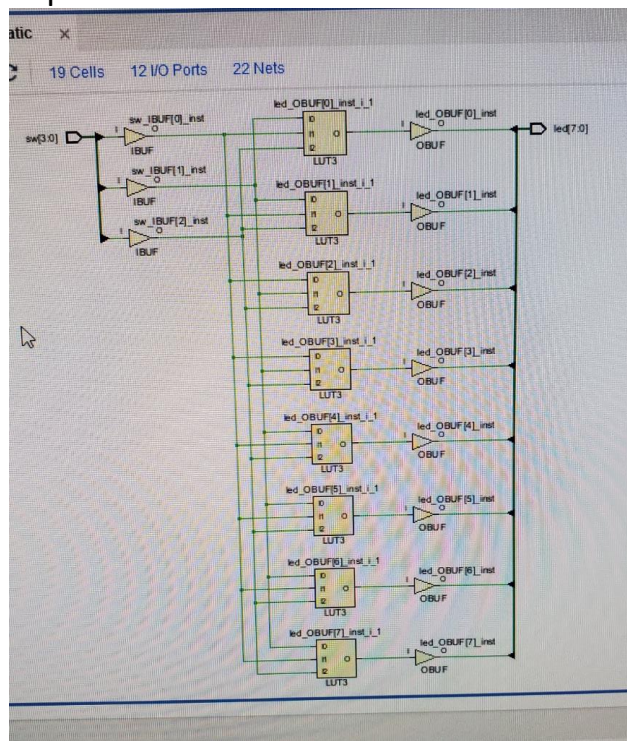
On the right, the structural schematic is shown. It illustrates the implementation of the decoder using LUTs (Look-Up Tables) and OBUFs (Output Buffers). The input 'sw' is connected to three LUTs, which then drive the output register 'led' through OBUFs. The schematic shows the internal logic of the decoder, including the LUTs and the output buffers.

At the bottom of the image, a table is visible, likely representing the results of a simulation or synthesis. The table has the following columns: TPWS, Total Power, Failed Routes, Methodology, and RQA Score. The table is currently empty, with only the headers visible.

TPWS	Total Power	Failed Routes	Methodology	RQA Score
------	-------------	---------------	-------------	-----------

Part 2 (Second Lab Day). Create a new decoder project with a different name and implement the same design using structural Verilog (gate-level) design techniques.

- A. Open the schematic view in Vivado, and study/understand the implementation.



- B. Use “or, and”, and other structural Verilog statements to implement the Decoder.

```
21 :
22 :
23 : module ANDOR(
24 :     input[3:0] sw,
25 :     output [7:0] led
26 : );
27 :
28 :
29 :
30 :     wire nota, notb, notc;
31 :     wire a,b,c;
32 :
33 :
34 :     not (nota,sw[0]);
35 :     not (notb,sw[1]);
36 :     not (notc,sw[2]);
37 :     and (led[0], nota,notb,notc);
38 :     and (led[1], nota,notb,sw[2]);
39 :     and (led[2], nota,sw[1],notc);
40 :     and (led[3], nota,sw[1],sw[2]);
41 :     and (led[4], sw[0],notb,notc);
42 :     and (led[5], sw[0],notb,sw[2]);
43 :     and (led[6], sw[0],sw[1],notc);
44 :     and (led[7], sw[0],sw[1],sw[2]);
45 :
46 :
```

```
module ANDOR(
    input[3:0] sw,
    output [7:0] led
);
```

```
    wire nota, notb, notc; // wires after the not gates
```

```
//the nots gates themselves getting an input from the sw bus
    not (nota,sw[0]);
    not (notb,sw[1]);
    not (notc,sw[2]);
```

```
    and (led[0], nota,notb,notc);
    and (led[1], nota,notb,sw[2]);
    and (led[2], nota,sw[1],notc);
    and (led[3], nota,sw[1],sw[2]);
    and (led[4], sw[0],notb,notc);
    and (led[5], sw[0],notb,sw[2]);
    and (led[6], sw[0],sw[1],notc);
    and (led[7], sw[0],sw[1],sw[2]);
```

```
endmodule
```

Conclusion:

In this lab, we used Vivado to program an FPGA in Verilog code to behave as a decoder. We did this twice in two different ways. From part 1, we did it using behavioral and in part 2 we did structural. Both of which behaved, as we had hoped, like a decoder. Both ways were 100% functional. We tested this using the switches and saw the correct output lights light up for a given input switch combination.