ECE 3300L

Lab 1

Instructor: Mohamed Aly

Group A

Sami Elias	012546378	samielias@cpp.edu
Joseph Popoviciu	014357772	jdpopoviciu@cpp.edu

Objective:

The objective of Lab 1 is to become more familiar with using Vivado software with Verilog code by creating the code for a 2x4 bit decoder, 3x8 bit decoder up to an nbits decoder then use testbench to test the corner cases by observing the output in the behavior graph and finally test our Verilog code with the ARTYX A7 100T.

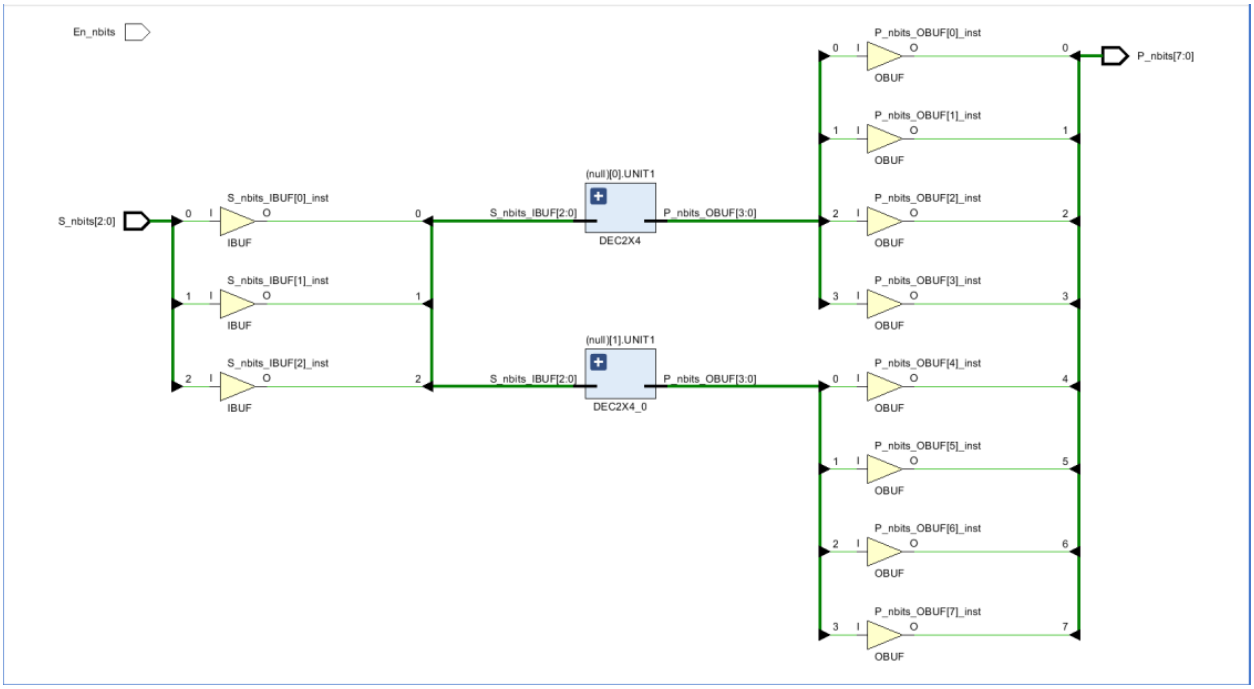

Figure 1

After writing our code we test to check if it passes synthesis. If it passes without error we are able to view the schematic that our code created. The image above is the schematic for a 3x8 decoder and was provided by Verilog after running synthesis. The 3 inputs on the left and the 8 inputs on the right indicate that it is a 3x8 decoder.

```
genvar i;
generate
if(DATA_WIDTH%2==1)//odd #ofinputs
begin
wire [1:0] temp_nbits;
assign temp_nbits[0]=~S_nbits[DATA_WIDTH-1];
assign temp_nbits[1]=S_nbits[DATA_WIDTH-1];
        for(i=0;i<(2**DATA_WIDTH)/4;i=i+1)
            begin
             DEC2X4 UNIT1
             (
                 .S(S_nbits[1:0]),
                 .P(P_nbits[4*i+3:4*i]),
                 .En(temp_nbits[i])
```

Figure 2
This is the piece of code that used the generate function to help create the decoder.

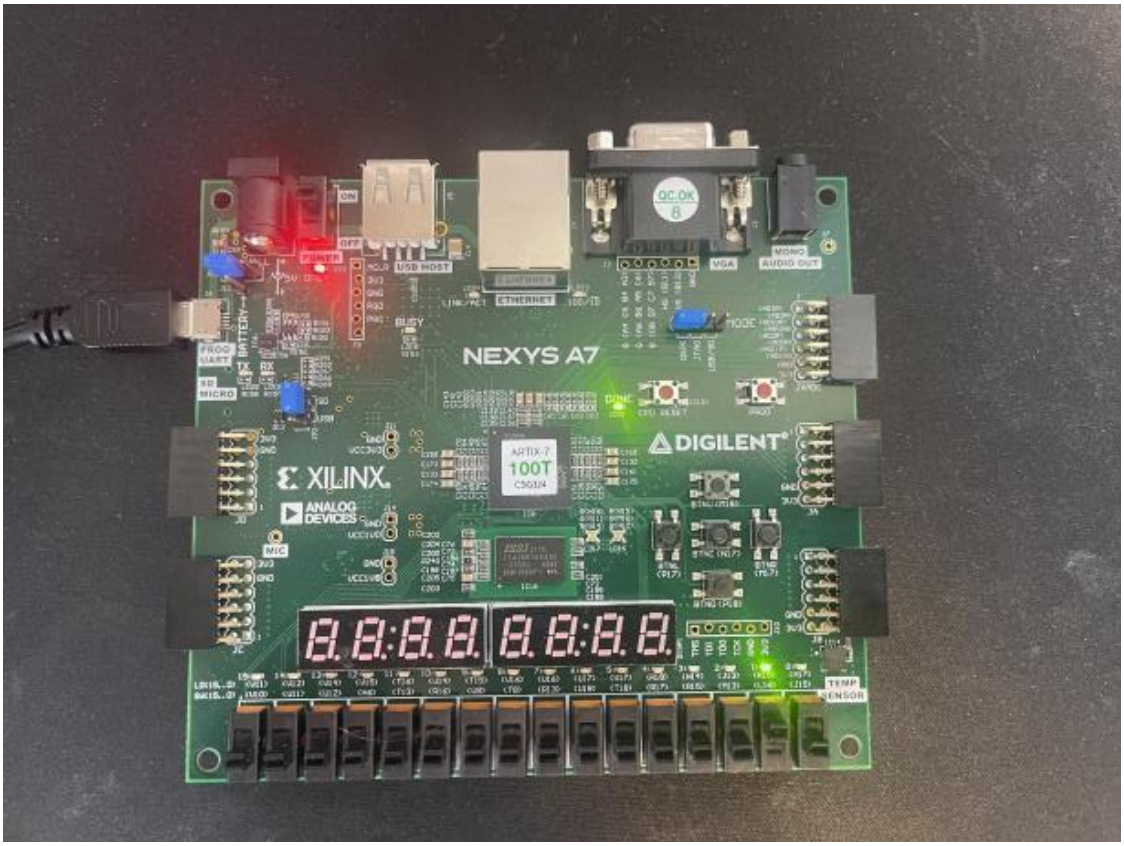

Figure 3

Figure 3 displays the Nexys A7 FPGA board which we use to physically test our Verilog code after running bit generation and programming the board. The 2x4 decoder is running bits 01.

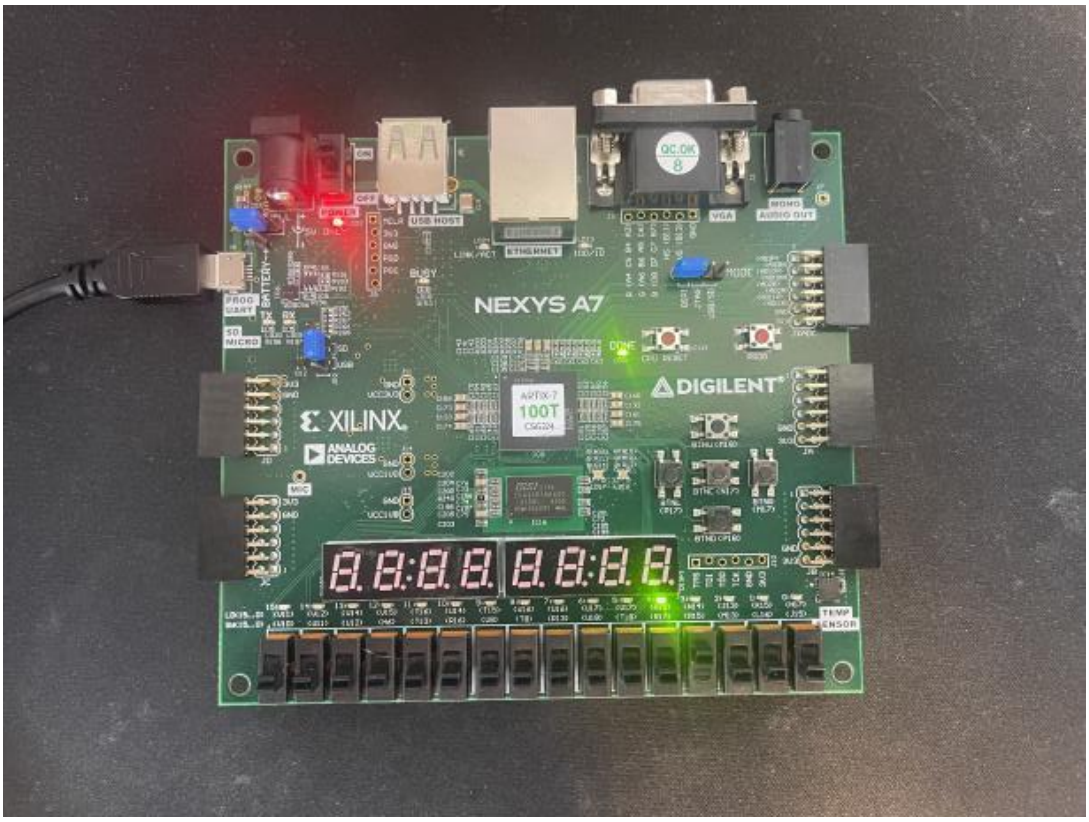

Figure 4

This is the 3x8 decoder running the value 100.

Test bench:

Figure's 5, 6 and 7 are the behavioral graphs of our code, this is what we use to check the corner cases for our output and was my primary method to debug my code and attempt to correct any errors. For example, in figure 7, the red blocks indicate a missing connection. Furthermore, our code did not work for a 5bit decoder. Initially I attempted to create an nbit decoder using one for loop, unfortunately once I had it working for a 3x8 decoder, it no longer worked for a 2x4 decoder. As a result, I searched resorted to learning how to use if statements within Verilog. We ultimately settled with if and else if statements separated our loops and decoders based on the nbits input modulus 2. This tells allows us to tailer our code to be able to handle even and odd bit inputs. Unfortunately, the code still did not work for a 5 bit and my only hypothesis is that I did not have enough input test bits instantiated within the test bench.

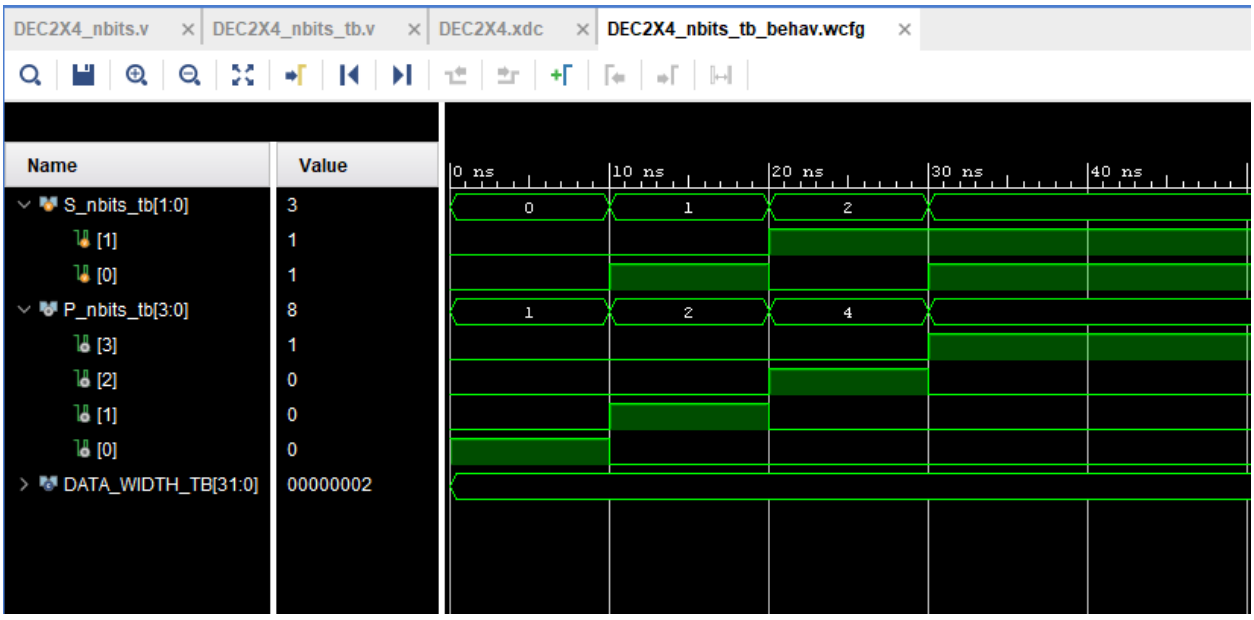

Figure 5

Figure 6

Figure 7

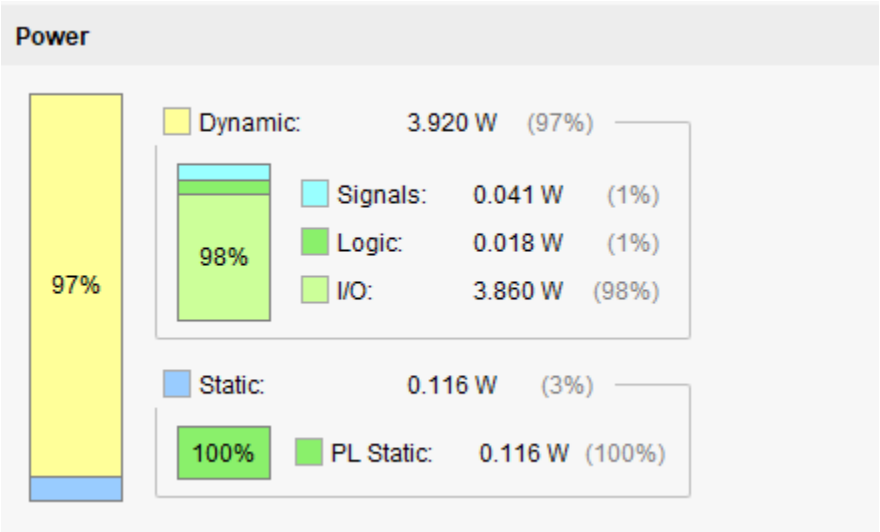

Figure 8

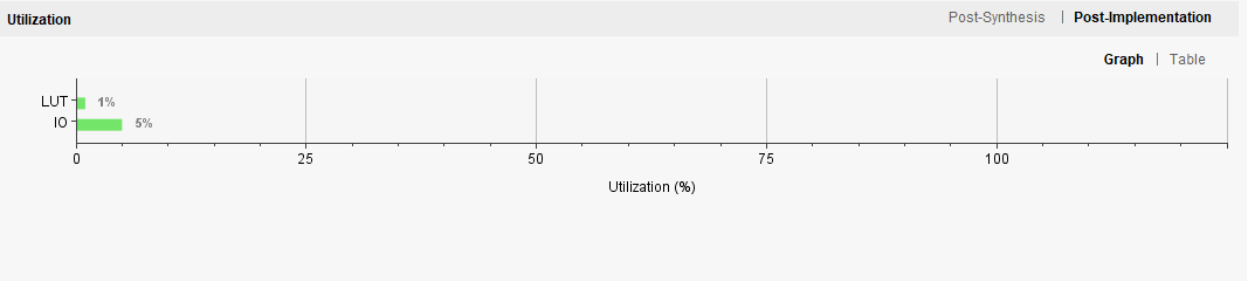

Figure 9

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP
synth_1	constrs_1	synth_design Complete!								4	0	0.00	0	0
impl_1	constrs_1	write_bitstream Complete!	NA	NA	NA	NA	NA	4.035	0	4	0	0.00	0	0

Figure 10

Figures 8, 9 and 10 are tools used to analyze how much of your resources are being used to run the code. Since we are using Verilog to create code that is used to program a hardware device it will naturally have constrains. For this nbits Lab our FPGA used a total of 3.920 W of power.

Conclusion:

We created a Verilog code that can run a 2x4 decoder and a 3x8 decoder. We had 3 inputs that were connected to two 2x4 decoders which ended up outputting 8 results. We used the test bench to help diagnose some problems before we ran it on the board. The main function we used to make the decoder was the generate function.