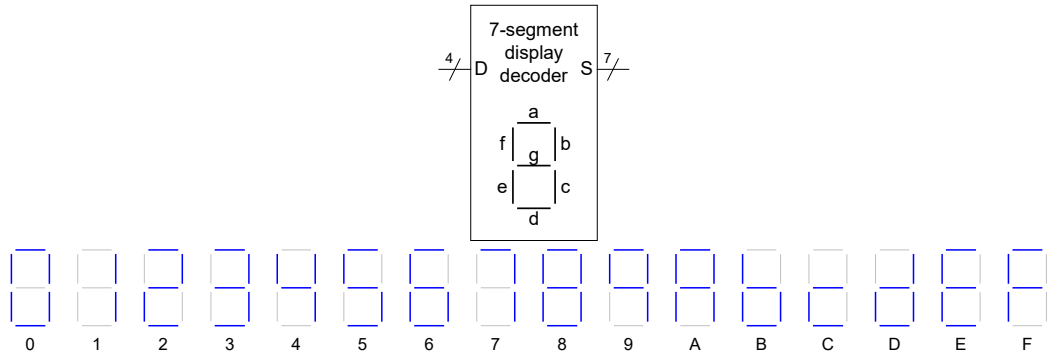
Lab 2: Seven-Segment Display Decoder

Authors: Andres Samson

Documentation: I received help from C3C Zack Duessler debugging my testbench and C3C Jake Marbach figuring out the error with my bitstream generation

Purpose: The purpose of this lab is to implement a seven-segment display decoder on the Basys3 development board using VHDL. The decoder will receive a four-bit binary input value and will display the equivalent hexadecimal value on the seven-segment displays on the Basys3 board. This design will make it convenient to quickly convert values from binary to the hexadecimal numbering system.

Prelab: In the pre-lab, I determined the necessary Boolean logic to implement the described design. I did this by determining which of the segments would need to be activated (turned high) in order to display the correct hexadecimal equivalent. My findings are compiled in Table 1 below.

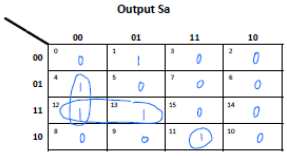


Hexadecimal Digit	Inputs				Outputs							(in hex)
	D3	D2	D1	D0	Sg	Sf	Se	Sd	Sc	Sb	Sa	
0	0	0	0	0	1	0	0	0	0	0	0	0x40
1	0	0	0	1	1	1	1	1	0	0	1	0x79
2	0	0	1	0	0	1	0	0	1	0	0	0x24
3	0	0	1	1	0	1	1	0	0	0	0	0x30
4	0	1	0	0	0	0	1	1	0	0	1	0x19
5	0	1	0	1	0	0	1	0	0	1	0	0x12
6	0	1	1	0	0	0	0	0	0	1	0	0x2
7	0	1	1	1	1	1	1	1	0	0	0	0x78
8	1	0	0	0	0	0	0	0	0	0	0	0x0
9	1	0	0	1	0	0	1	1	0	0	0	0x18
A	1	0	1	0	0	0	0	1	0	0	0	0x8
B	1	0	1	1	0	0	0	0	0	1	1	0x3
C	1	1	0	0	0	1	0	0	1	1	1	0x27
D	1	1	0	1	0	1	0	0	0	0	1	0x21
E	1	1	1	0	0	0	0	0	1	1	0	0x6
F	1	1	1	1	0	0	0	1	1	1	0	0x0E

Table 1 – Showing the mapping between possible inputs and desired outputs for each segment

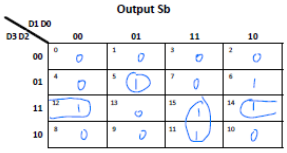
Utilizing Table 1 above, I derived the Boolean equations that would represent the desired outputs. I utilized K-maps to streamline this process, as seen below.

Output Sa



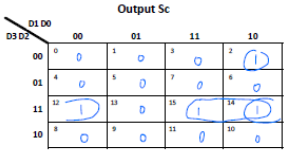
Equation: $S_a = D3\,D2\,D1' + D3'D2'D1'D0 + D3\,D2'\,D1\,D0 + D2\,D1'D0'$

Output Sb



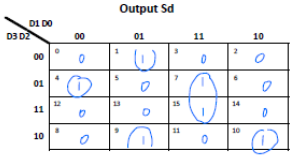
Equation: $S_b = D3\,D2\,D0' + D3\,D1\,D0 + D2\,D1\,D0' + D3'D2\,D1'D0$

Output Sc



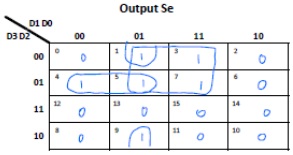
Equation: $S_c = D3\,D2\,D0' + D3\,D2\,D1 + D3'D2'\,D1\,D0'$

Output Sd



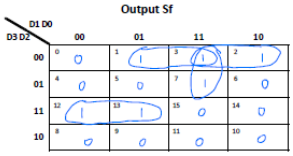
Equation: $S_d = D3'D2\,D1'D0' + D2'D0\,D1' + D3D2'D1\,D0' + D2D1D0$

Output Se



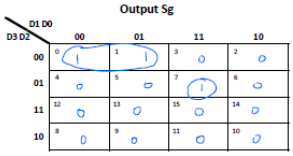
Equation: $S_e = D3'D2D1' + D2'D1'D0 + D3'D0$

Output Sf



Equation: $D3\,D2D1' + D3'D2'D0 + D3'D2'D1 + D3'D1D0$

Output Sg



Equation: $S_g = D3'D2D1D0 + D2'D1'D3'$

Design: The final design for the implementation of this lab is show below in Figure 1.

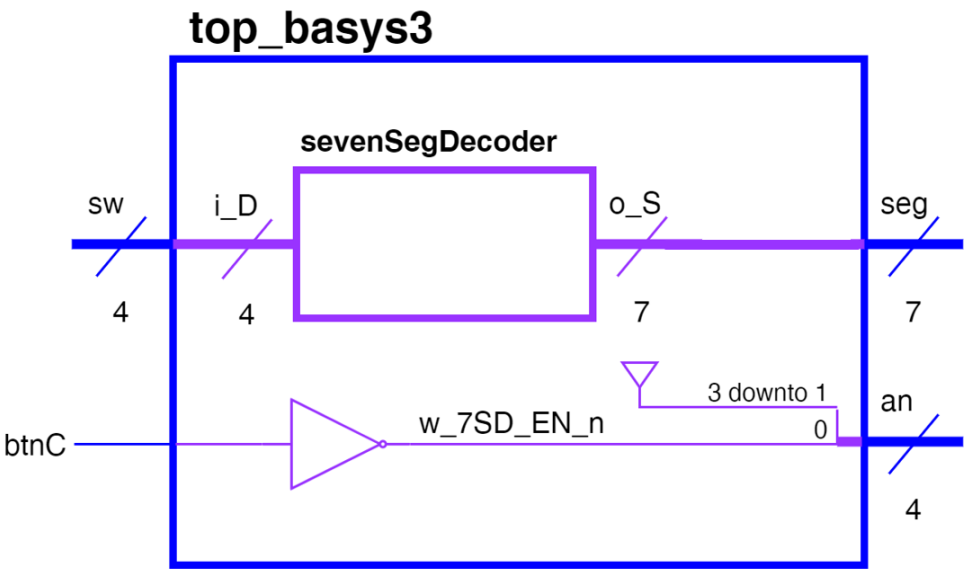


Figure 1. Top Level Schematic

Our first step in reaching this goal was to design and test the SevenSegDecoder component separately utilizing the equations Sa-Sg above and matching Figure 2.

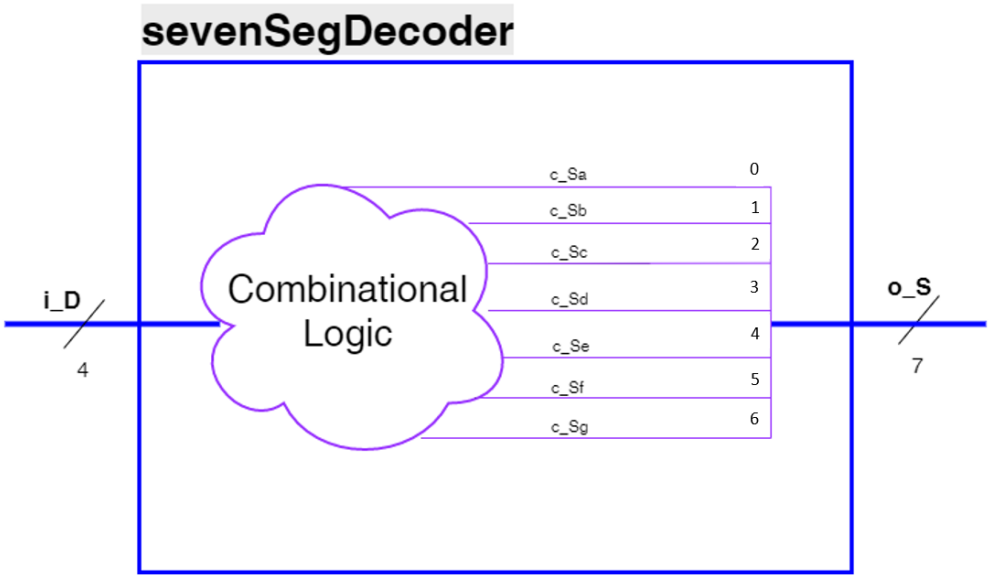


Figure 2. High Level Approach View of sevenSegDecoder

I started by setting up the component file utilizing the provided class template and renaming it SevenSegmentDecoder. I defined the entity to match Figure 1 with a four-bit i_D input and a seven-bit o_S output. I then created another signal vector with 7 values to represent Sa-Sg in the logic equations. I initialized all of these values to 1. I then implemented each of the equations, using both the Boolean expressions and a LUT for at least two of the first four equations. I found that I preferred using the Boolean expressions since searching through the table for the high outputs was very time consuming, and it was a lot more intuitive to be able to see the equations and immediately understand what they meant and what they were supposed to represent.

The next step was to design the test bench to ensure the component worked correctly. I modified the class testbench template and renamed it sevenSegmentDecoder_tb.vhd. I then declared the sevenSegmentDecoder and instantiated a version of it. I mapped the component to the artificially created input sw and output w_seg. I used assert statements to ensure that the outputs from each of the inputs were what I expected. I had a lot of trouble implementing this test bench. The program would fail on the input "0x6" every time. I double checked all of my code, ensuring that my connections were correct, and I had typed all of my equations properly. After conferring with C3C Zack Duessler, he suggested that I change my assert statements to be in binary and everything ran as expected after that. The simulation waveform can be seen in Figure 3.

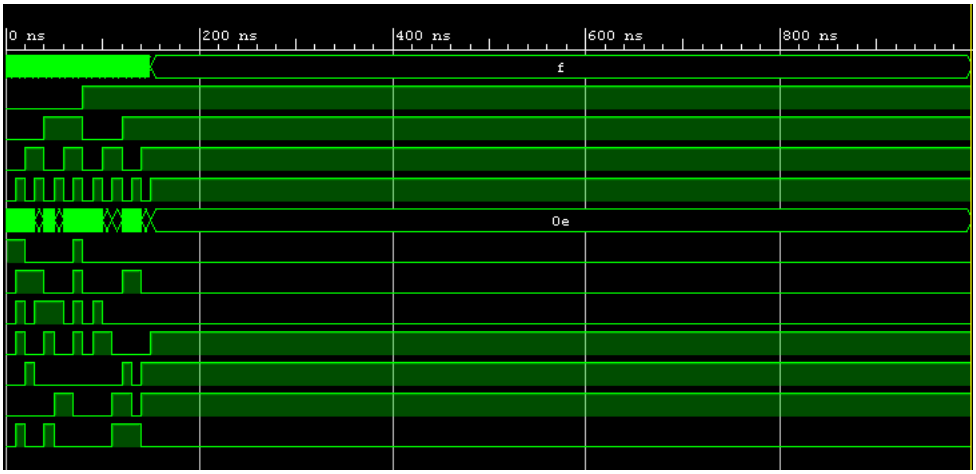


Figure 3. Simulation Waveform

I examined the waveform and confirmed that test cases 0x0 – 0xF were all run. For each test case, I checked the output vector's hexadecimal values against the values we determined in Table 1.

The next design step was to create the top_level design file to match Figure 1. I started with the provided `top_basys3.vhd` file provided for the lab. The entity was already defined for me, so I just needed to complete the architecture. I declared the `sevenSegmentDecoder` and created a wire `w_7SD_EN_n` for the button based enable. I then instantiated the component and connected `i_D` to `sw` and `o_S` to `seg`. I then connected `w_7SD_EN_n` to `not btnC` (since the an is 0 enabled) and connected `an(0)` to `w_7SD_EN_n`. The remaining three bits of an were connected to '1'. I implemented this all correctly the first time through.

Final Results: Once I finished creating the design in VHDL looked at the RTL schematic for both the top_basys3 design and the sevenSegmentDecoder design. Figure 4 shows top_basys3.

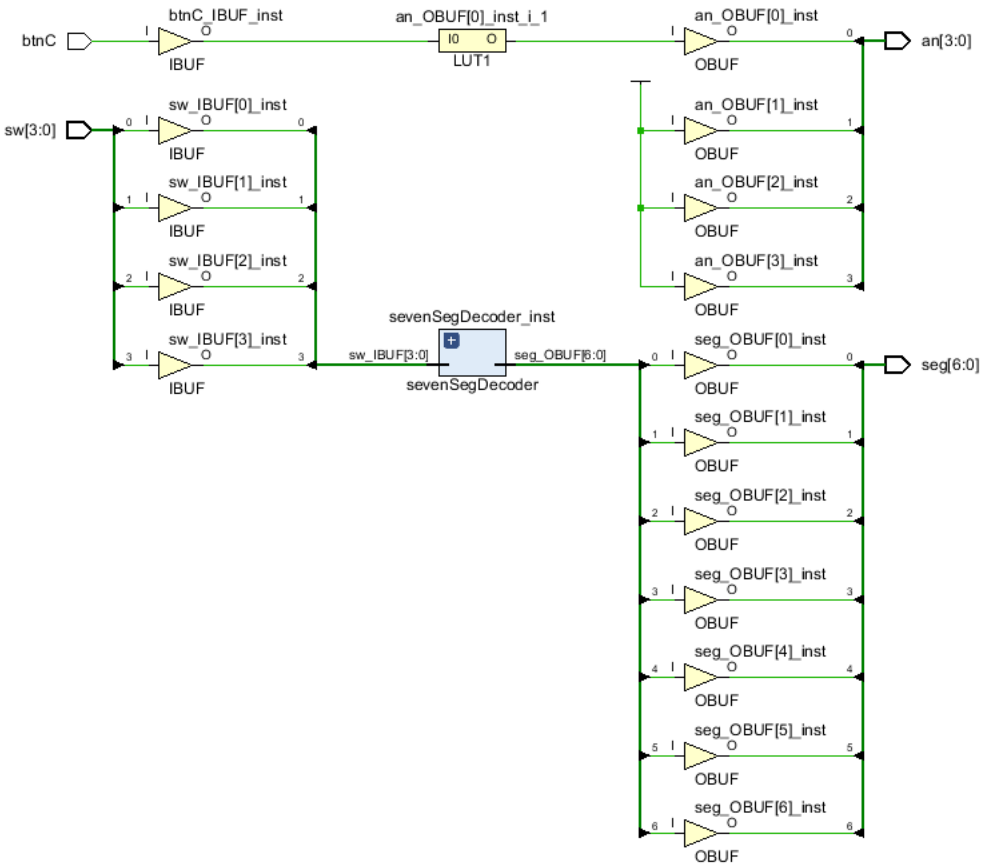


Figure 4. Top_Basys3 RTL Schematic

Figure 5 shows sevenSegmentDecoder.

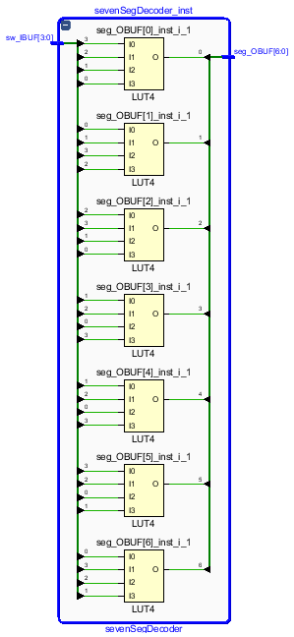


Figure 5. SevenSegDecoder RTL Schematic

The final step was to generate the bitstream for the project. Because the top_basys3 utilized standard notation for sw and seg, I only had to uncomment the respective lines in the Basys3_Master.xdc file. The bitstream generation failed initially because I had not uncommented some of the necessary lines. Then it kept failing and after talking to C3C Jake Marbach, I determined that in the settings, the project device had not been set to the Basys 3 board, so I had to do this manually. Finally, I pushed the generated bit stream to the board and successfully demonstrated all 16 possible inputs. The demonstration video is included in the README file.

Conclusions: I think this was a very cool lab because it was more challenging than anything else we have done before, and it was fun thinking through a more complex problem. Seeing your design actually work on the board is very satisfying and it affirms my love for seeing digital design be implemented in a physical medium. I became a lot more comfortable and knowledgeable with VHDL after completing this lab, so that is another big advantage.

Reflection:

- **Number of hours spent on Lab2 (Combined):** ~10
- Figuring out how to create the necessary structures was very difficult since I do not really understand VHDL. I looked back at other assignments and pieced it together after a very long time. I also had a lot of trouble getting the test bench to work and my bitstream generation to happen successfully. I asked some of my friends for help with these problems.
- I looked at Lab 1 and Ice 3 and heavily relied on these examples
- It would be nice to have a more substantial lesson for teaching us how to use VHDL. I still do not think I completely understand how the language works. For example, if I did not have old code to reference, I would have no idea how to complete these labs. This is frustrating because when I go to debug my code, I have no idea where to start since I'm not sure how it's supposed to work.