

Lab 2
Simon Carballo
1618309
10/23/2020
Section D

Write Up

- **Description:** In this lab we were asked to create a functional 7-segment Display on the Basys3 board that can change values using seven switches. In order to create this, we were tasked with designing a 4-bit adder, and a seven-segment-converter to put together using wires.
- **Design:** For this design we have the Top Level, with 2 subsections including the 4-bit adder and the seven-segment converter. Inside of the 4-bit adders are 3 Full adders ($n\text{-bit} = (n-1)\text{Full_Adders}$) that change the input of the converter depending on the active switches.

- 4-bit Adder:

First let's look at a normal Full Adder:

- Truth Table of a Full Adder:

a	b	c	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

- As we see in the truth table we add the sum of a, b, and cin, and when there is a value over 1, there would be an overflow representing Cout.
- We get this equation for the Sum and Cout according to the truth table:
 - $$\begin{aligned}\text{Sum} &= !a!bc + !ab!c + a!b!c + abc \\ &\quad !c(!ab + a!b) + c(!a!b + ab) \\ &\quad !c(a \text{ xor } b) + c(a \text{ xand } b) \\ &\quad !c(x) + c(1) \\ &\quad c \wedge x\end{aligned}$$

$$c \wedge a \wedge b$$

Thus with Simplification we get: $\text{Sum} = c \wedge a \wedge b$

$$\text{Cout} = !abc + a!bc + ab!c + abc$$

$$c(!ab + a!b) + ab(!c + c)$$

$$c(a \wedge b) + ab$$

Thus with Simplification we get: $\text{Cout} = c(a \wedge b) + ab$

- With this knowledge of how a Full Adder works we can create it in Verilog with the proper boolean expressions. In order to find how many Full Adders we will need we use the expression: $n\text{-bit} = (n-1)\text{Full_Adders}$.

- 4-Bit Adder:

$$4\text{-bit} = (4-1)\text{Full_Adders}$$

$$3\text{Full_Adders}$$

- By wiring the Cout of each adder into the next we create a ripple-carry adder with the sums being the outputs necessary for our next subsection (n_0, n_1, n_2, n_3).

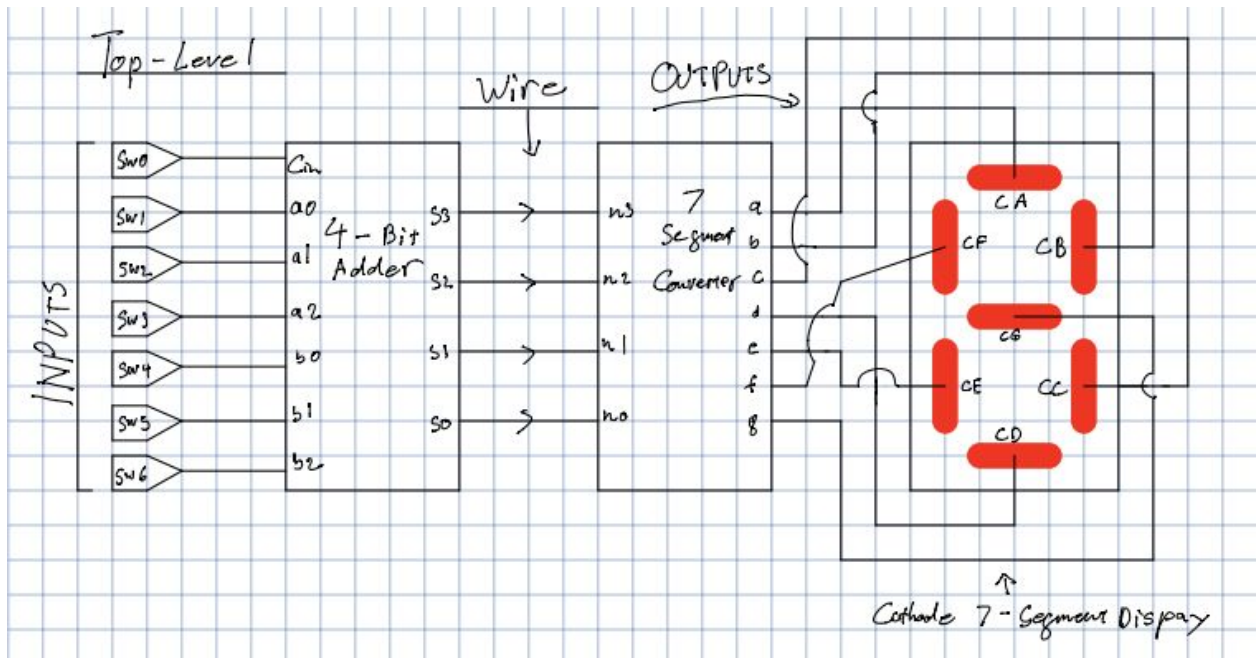
- Seven-Segment Converter:

First let's take a look at the truth table:

n3(d)	n2(c)	n1(b)	n0(a)	a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	1	1	1	1
0	0	1	0	0	0	1	0	0	1	0
0	0	1	1	0	0	0	0	1	1	0
0	1	0	0	1	0	0	1	1	0	0
0	1	0	1	0	1	0	0	1	0	0
0	1	1	0	0	1	0	0	0	0	0
0	1	1	1	0	0	0	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	1	1	0	0
1	0	1	0	0	0	0	1	0	0	0
1	0	1	1	1	1	0	0	0	0	0

1	1	0	0	0	1	1	0	0	0	1
1	1	0	1	1	0	0	0	0	1	0
1	1	1	0	0	1	1	0	0	0	0
1	1	1	1	0	1	1	1	0	0	0

- This is a 4-variable truth table for the 16hex values available on a seven segment display. CA - CG represents each segment of the Cathode Segment Display. There is a DP segment but that will be unused for this design. In order to figure out active high and lows of each segment we go through each hex value and determine whether the segment will be on or off. For example, when we want the value “1” we would need to have the segments ‘CB’ and ‘CC’ on therefore making them the lows(0s) for a cathode diode.
- With this truth table we can acquire a boolean expression for each segment to make the 16Hex values (n0,n1, n2, n3 == a, b, c, d):
 - $a = a!b!c!d + !abc!d + ab!cd + a!bcd$
 - $b = a!bc!d + !abc!d + ab!cd + !a!bcd + !abcd + abcd$
 - $c = !ab!c!d + !a!bcd + !abcd + abcd$
 - $d = a!b!c!d + !a!bc!d + abc!d + a!b!cd + !ab!cd + abcd$
 - $e = a!b!c!d + ab!c!d + !a!bc!d + a!bc!d + abc!d + a!b!cd$
 - $f = a!b!c!d + !ab!c!d + ab!c!d + abc!d + a!bcd$
 - $g = !a!b!c!d + a!b!c!d + abc!d + !a!bcd$
- We can input this boolean expression into the converter by taking in the 4 outputs and wiring each output to its respective segment. We will do this in the Top_Level Diagram as shown below.
- Top-Level:



- As you can see in the diagram, the Top-Level connects all the components we designed to enable the 7-segment display to properly function with the 7 switches as inputs.
- **Testing & Simulation:**

In order to test my design I used two different resources. The first resource is by running a simulation directly from the Vivado application. Using the provided testbench file, I entered in the 16 different cases spaced between each 100ns. By spacing out each test by 100ns, I can clearly distinguish each switch change made, and the corresponding highs and lows of the output. Another source was by programming the bitstream to the Basys3 board. This method is a more efficient method of testing, as there is not too much going on with the Top_Level and there are 7 switches to interact with to get different values.
- **Results:**
 - Document which pins of the FPGA you used and how they were connected to the switches and 7-segment displays.
 - As shown above in the top_level diagram I used switches sw[6:0] as the inputs to the adder and wires t[3:0] to connect the 4-bit adder to the 7-segment converter. The converter outputs CA - CG to the seven segment display. In order to activate the displays AN0 is set to 0, while AN1, 2, 3, and DP are set to 1.
 - Determine the longest path from any input to any output in your 3-bit adder. (i.e. the highest number of gates that any input goes through before reaching an output.)

- The longest path for an input to reach its output in a 4-bit adder is 9 logic gates.
- For an n -bit adder determine the length of the longest path from any input to any output (in terms of n).
 - Path = $n\text{-bit} * 3\text{gates}$
 - The longest path lies with the Cin which adds 3 gates (AND, OR, OR) per adder.
- Calculate the number of possible input values (7-bit vectors) for your adder? Give the percentage of these that you tested in your simulation.
 - In a 7-bit vector there are 128 possible input values/combinations which stems from the equation 2^n -bit
 - Since we only did 15 tests $(15/128)*100 = 11.72\%$ of tests were conducted on the simulation
- **Conclusion**
 - From this lab I learned how to manage the hierarchy in Verilog, which helped with designing the Top_Level by only calling components to connect together with wires. I learned how to create n -bit ripple carry adders, and how to create a seven segment converter. If I was to do this lab again, I would definitely set a better schedule for myself. I ended up turning the lab in one day late because I didn't ask enough questions on the Tuesday section. This caused me to get stuck on understanding how to use xilinx rather than understanding the main design of this lab. Although, I turned the lab in one day late, I definitely had very cleanly written code and would only need to use Buses to optimize this design.