CMPE 100 Lab Report 2

Leonid Shuster

Lab Section 1C

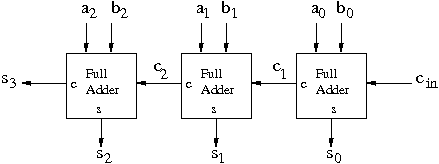
4/19/19

**Description:**

In this lab we were tasked with designing a 3-bit ripple-carry adder that displayed its result on one of the 7-segment LED displays on the Basys3 board. The ripple-carry adder added two 3-bit vectors corresponding to switches 0 - 6 on the board, and displayed the 3-bit sum and final carry-out bit vector as a hex digit on the right-most 7-segment LED display. Through this lab, we learned about using hierarchy of modules in our design, and also how to use the Vivado simulator to simulate our design. The lab consisted of the following three parts.

**Part One: Adder Design**

In the first part of the lab we learned about the hierarchy of modules by creating a ripple-carry adder that added two 3-bit vectors by using three full adders, with each full adder adding two respective bits from both vectors. The following schematic represents the ripple-carry adder:



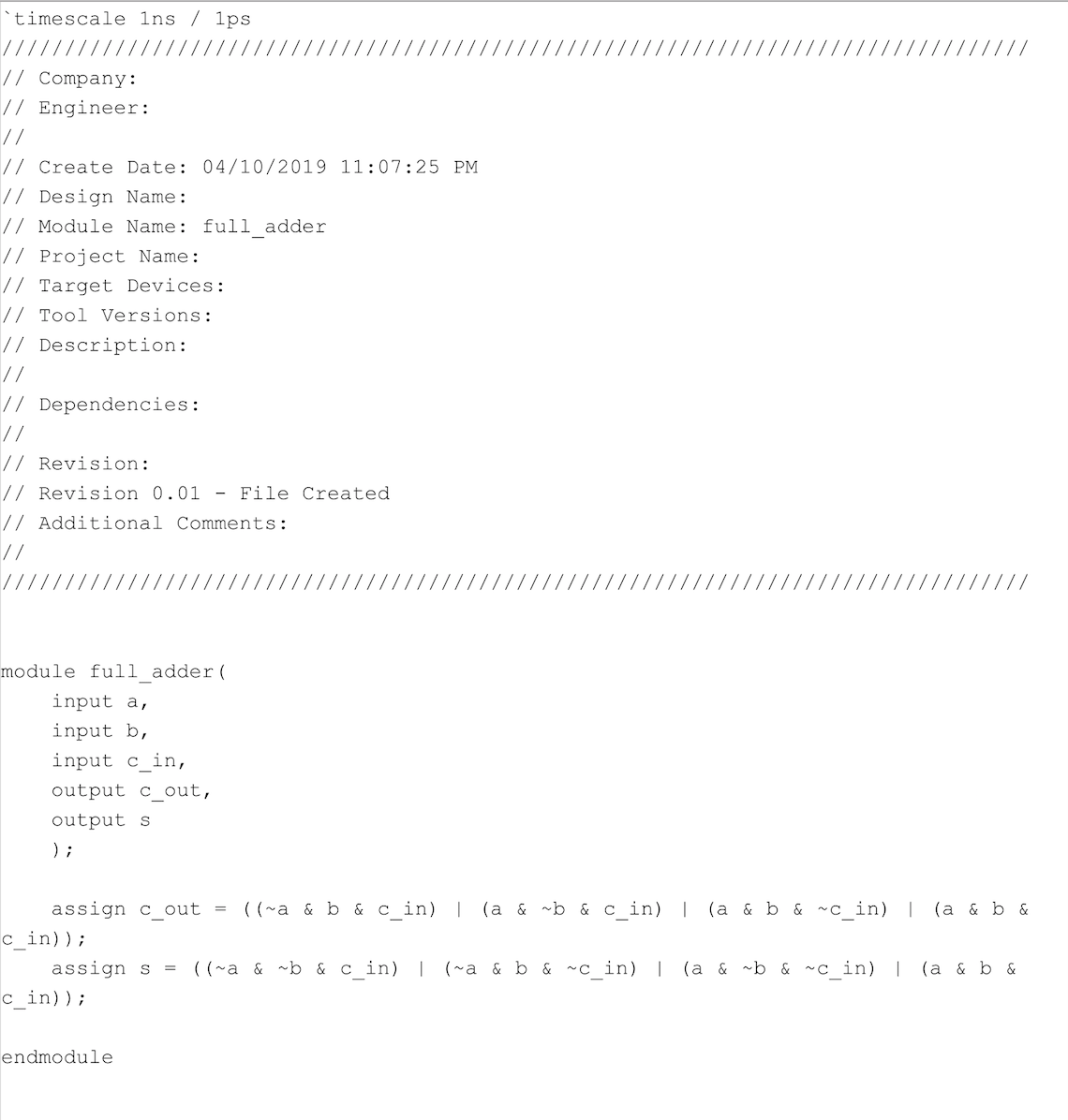
Ripple-carry adder (from CMPE 100 website)

First we had to figure out logic for a single full adder. A full adder combines three bits (a, b, cin) into two (cout, s). Following this logic, the following truth table was derived:

| **a** | **b** | **cin** | **cout** | **s** |
| --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

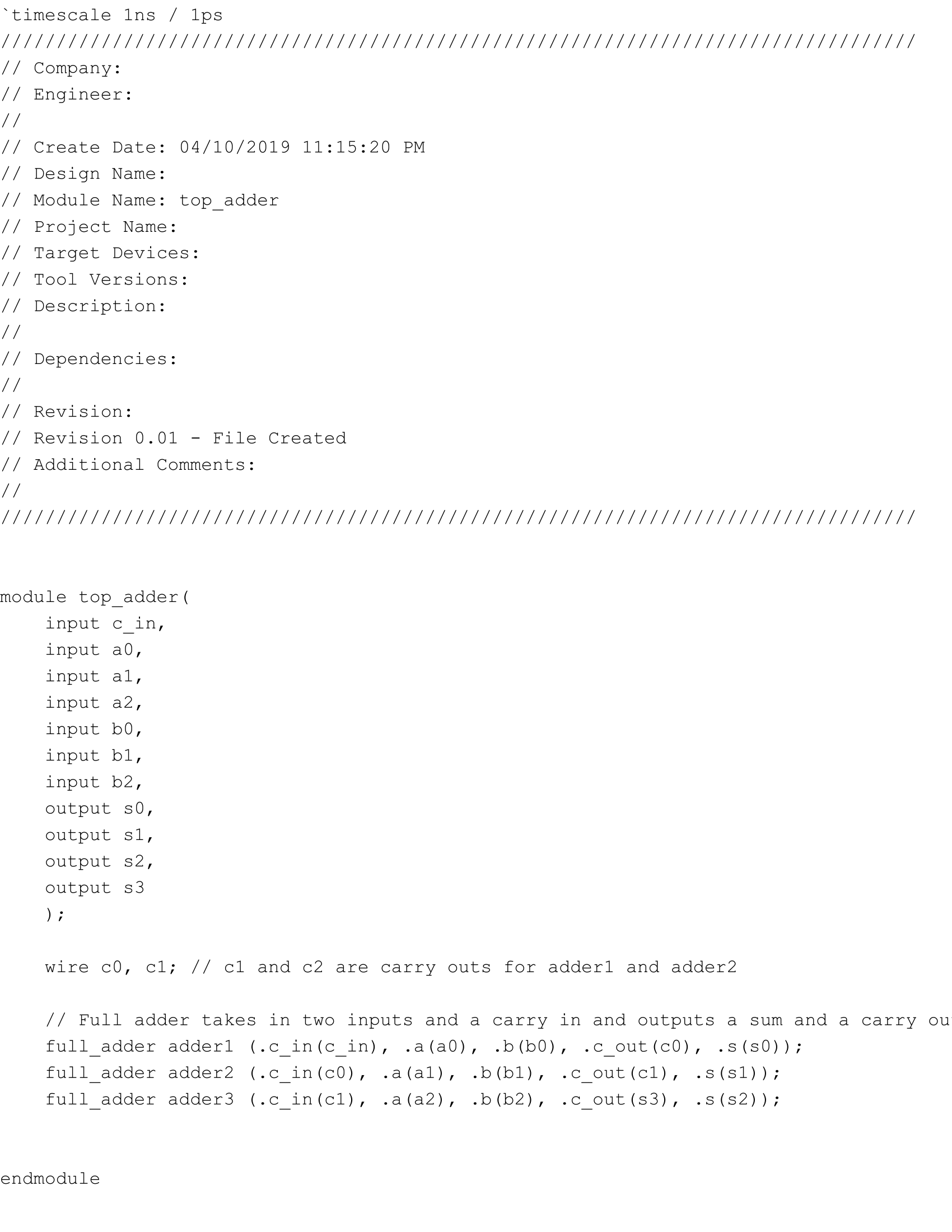
After creating the truth table, the following boolean expressions could be obtained:

I then implemented the design into a file named full\_adder:



full\_adder.v

The next step was to create the ripple-carry adder by putting together three full adders. In order to implement the ripple-carry adder, three full adders had to be used, which can be done in Verilog by creating three instances of the full\_adder module, with each taking in different bits of the 3-bit vector and a carry in, and outputting a sum and a carry out for the next adder. The design was implemented in top\_adder:



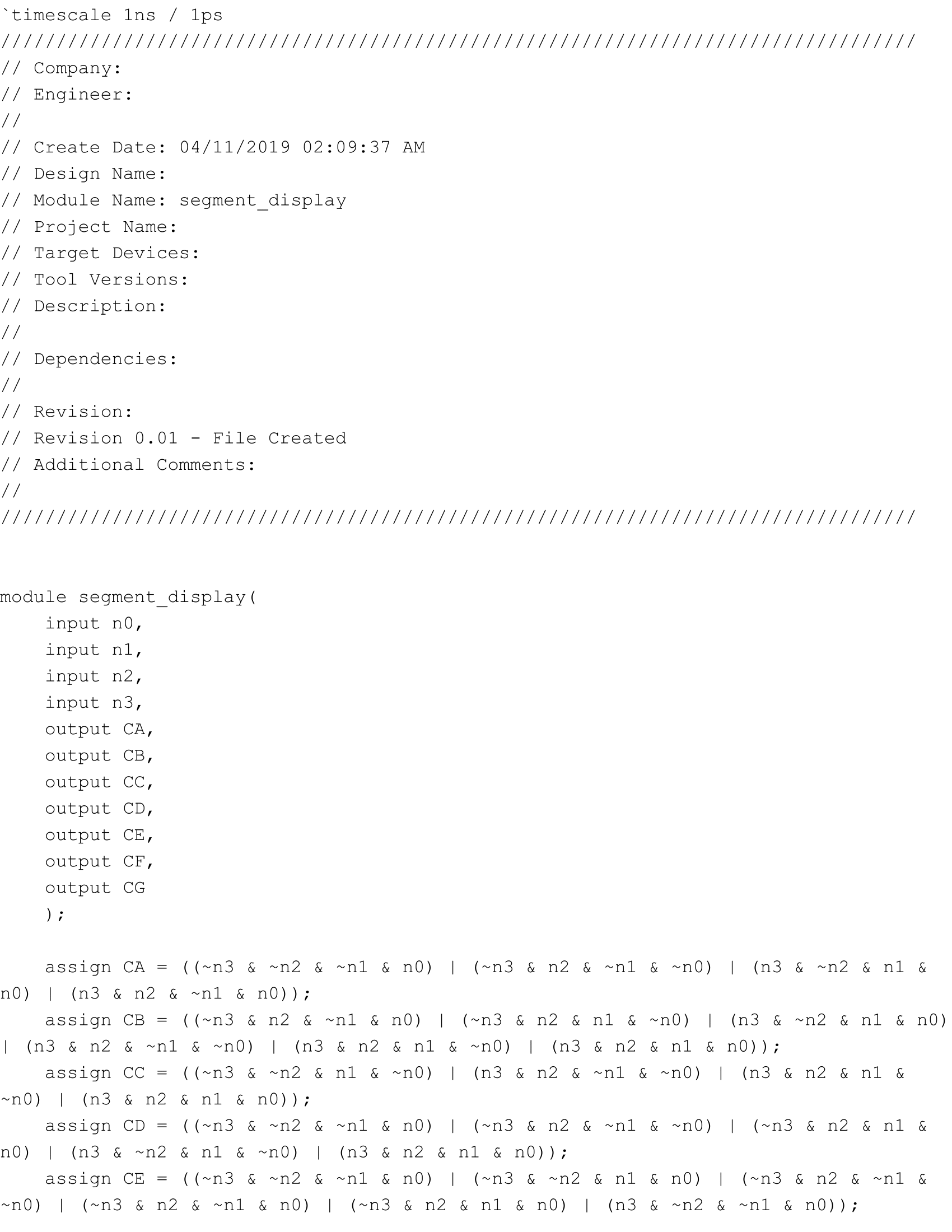
top\_adder.v

The longest path from any input to any output in my 3-bit adder is 3 if there continues to be a carry out through every adder.

The length of the longest path for a n-bit adder from any input to any output is n times if carry out continues to carry through every adder.

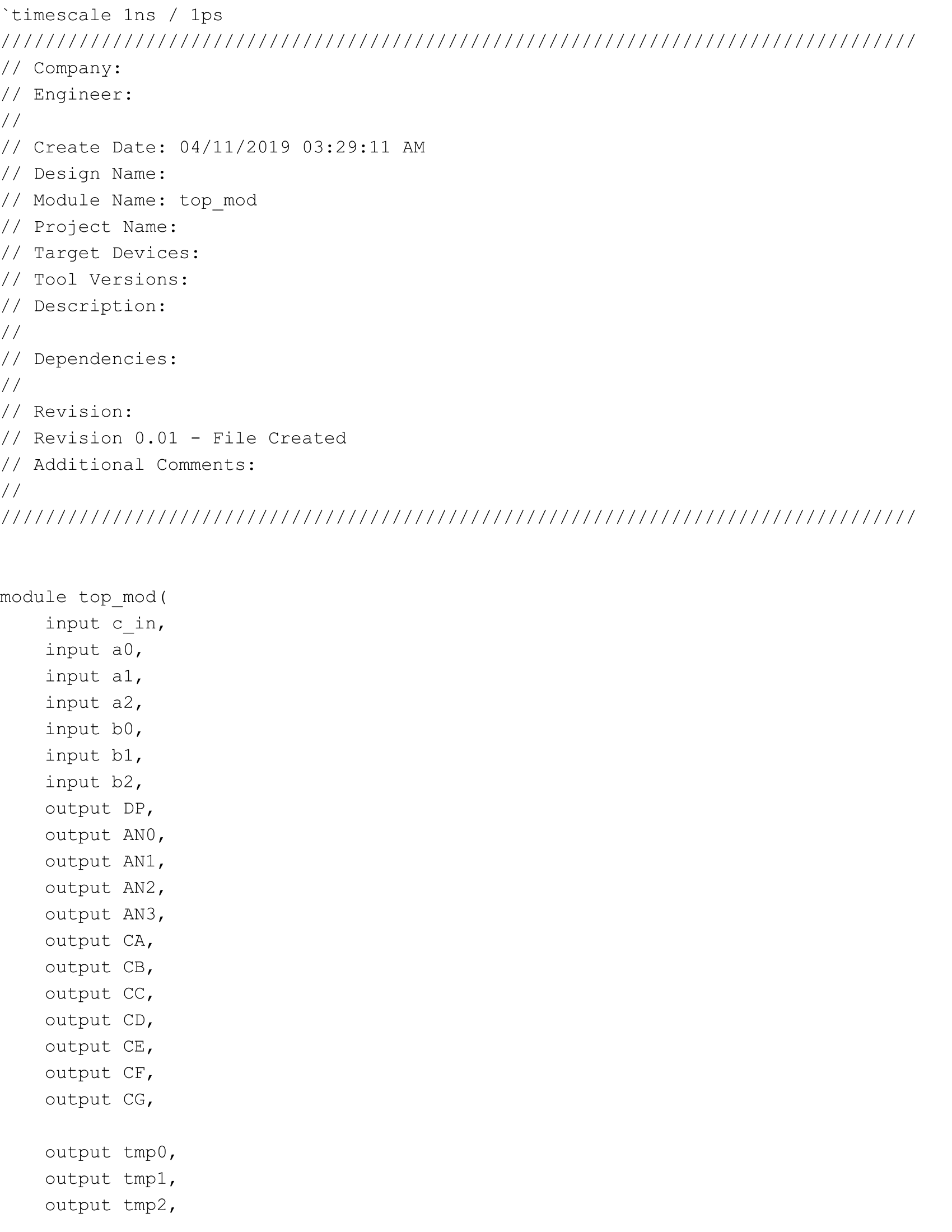
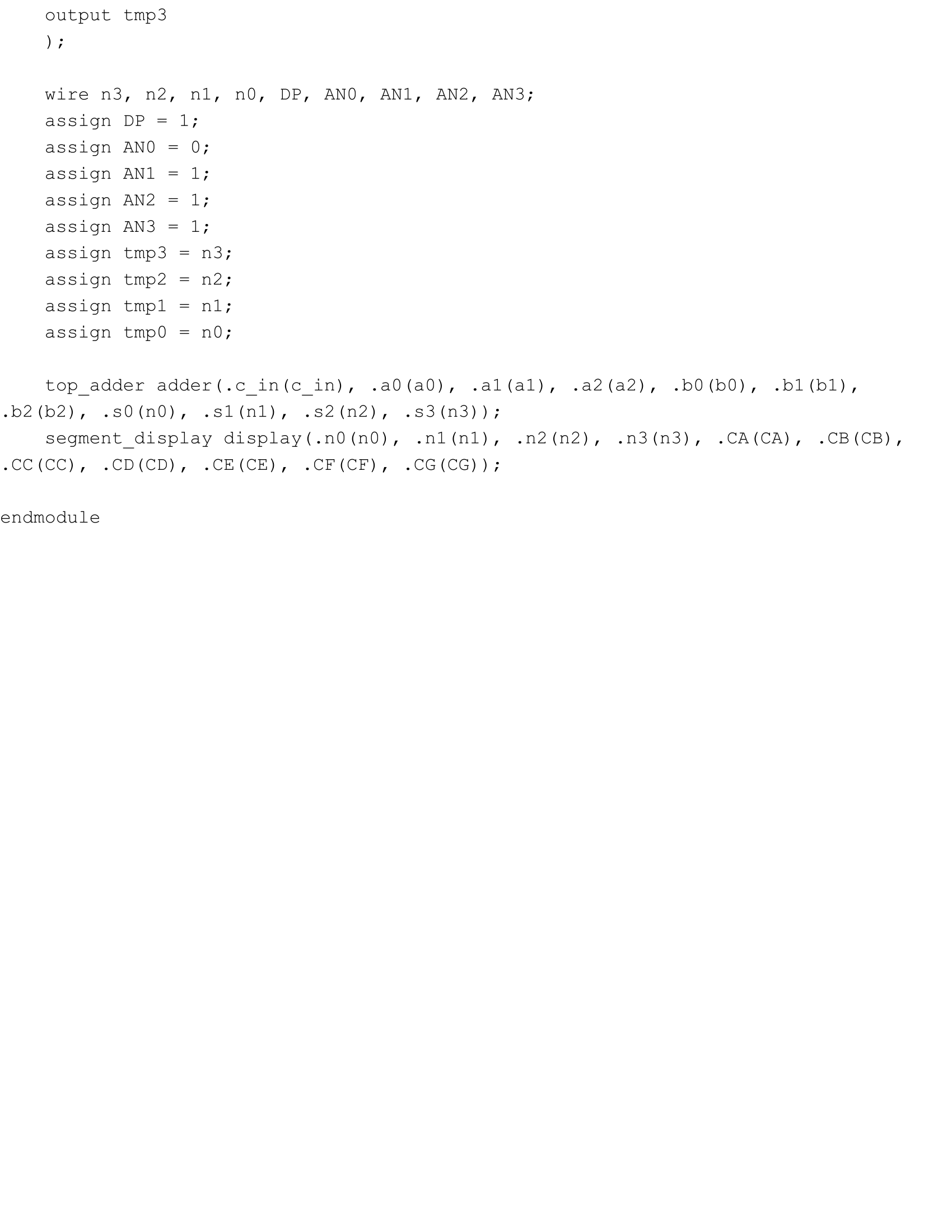
**Part Two: 7-Segment Display**

After successfully implementing a ripple-carry adder, the next step was to display the result of the 3-bit addition onto the rightmost 7-segment display as a hex number. In order to do so, the 4-bit vector (n3, n2, n1, n0) had to be converted to a 7-bit vector that would regulate the 7-segment LEDs in the display, where n0 was the sum of the first bits, n1 was the sum of the second bits, n2 was the sum of the third bits, and n3 was the final carry out. Each segment had a separate LED, and could be regulated by turning it off by giving it a high input, and turning it on by giving it a low input. First, a truth table had to be made by checking what LED segments should be on or off for every combination of bits and their sum. Then, boolean expressions could be derived for every LED segment. The design was implemented into segment\_display:

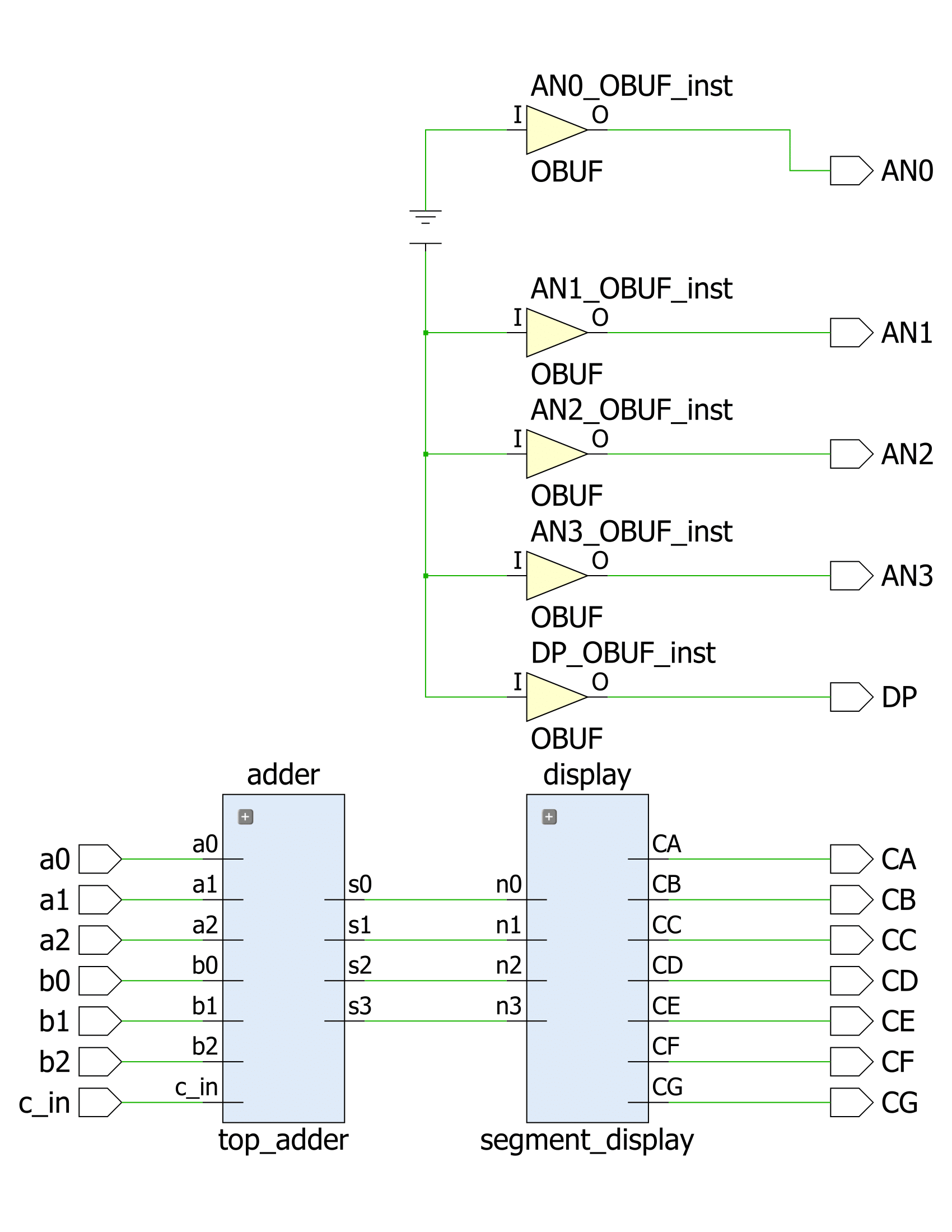
 

segment\_display.v

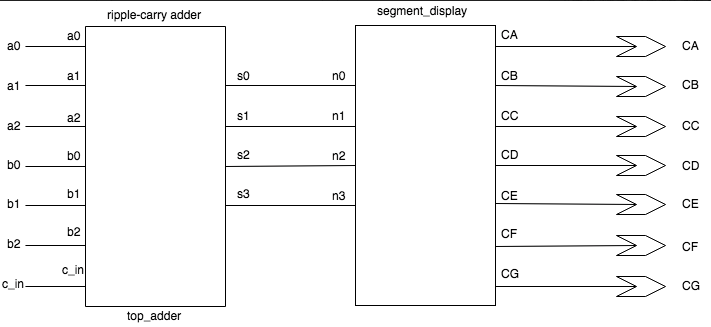
We then had to put everything together into a top level module. In a single file, two 3-bit vectors were fed into a ripple-carry adder that returned a 4-bit vector, which was then given as input to the 7-segment display that assigned values to the LED display and showed the sum in a hex number. The following are the code and schematic for the top level module:

top\_mod.v



top\_mod schematic (Xilinx)

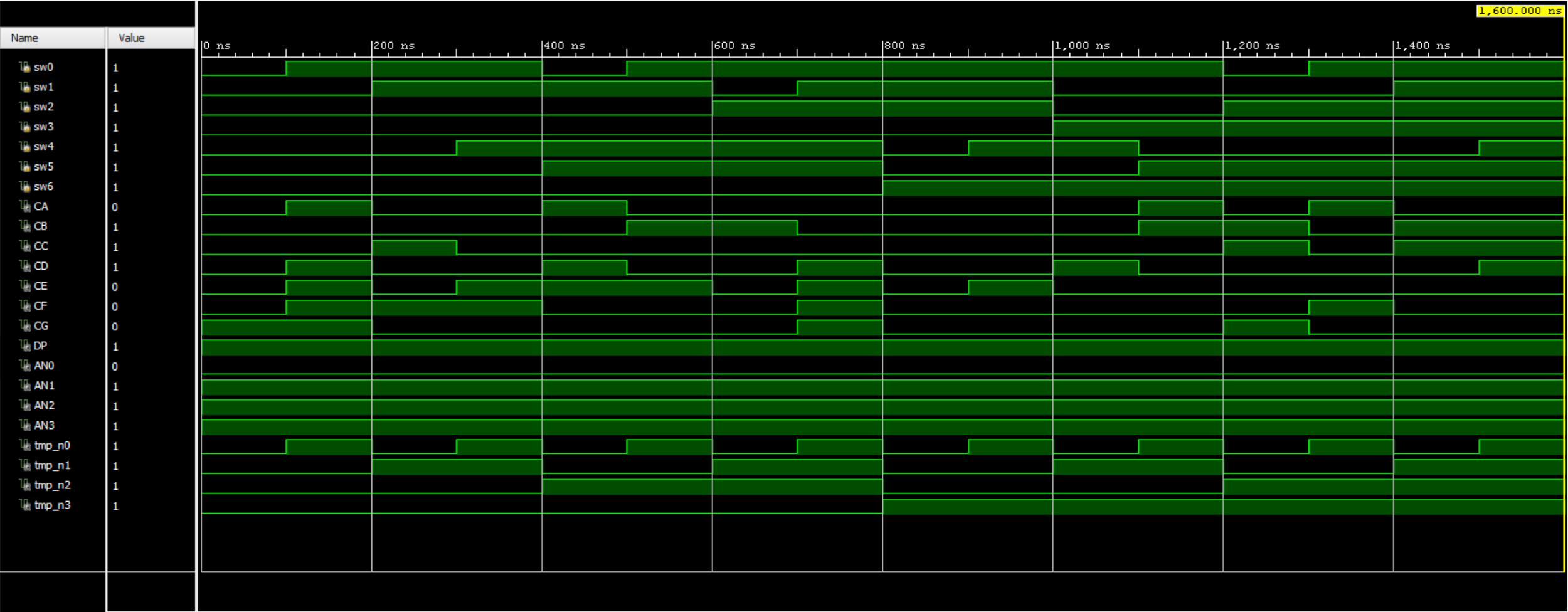


top\_mod schematic (Draw.io)

After mapping on the FPGA our two 3-bit vectors to switches 0 - 6, LED segments A - G to seg 0 - 6, an[0] to 0 indicating that the rightmost 7-segment LED be on, an[1], an[2], an[3], and dp to 1 indicating that the other LEDs and decimal be off, I tested out flipping the switches on and off and saw that the correct hex value was being displayed on the 7-segment LED display.

**Part Three: Using the Simulator**

In the last part of the lab, we learned how to use the simulator with a testbench that fed our top level module inputs and displayed waveforms modeling how it would be if the Basys3 board switches were really being flipped on and off. The sample inputs, which turned switches 0 - 6 on or off, only included inputs that produced sums up to 10, and we were tasked to fill in the remaining inputs needed to produce the remaining sums. I made a truth table for the remaining sums based on which switches should be on or off, added values for those switches on the testbench, and tested it on the simulator. The results are the following:



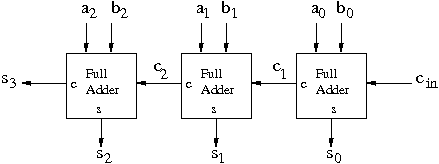
Waveform displaying all possible inputs and sums

The number of possible input values for the adder is = 128. I only tested out 16, which is 12.5% of the total possible values.

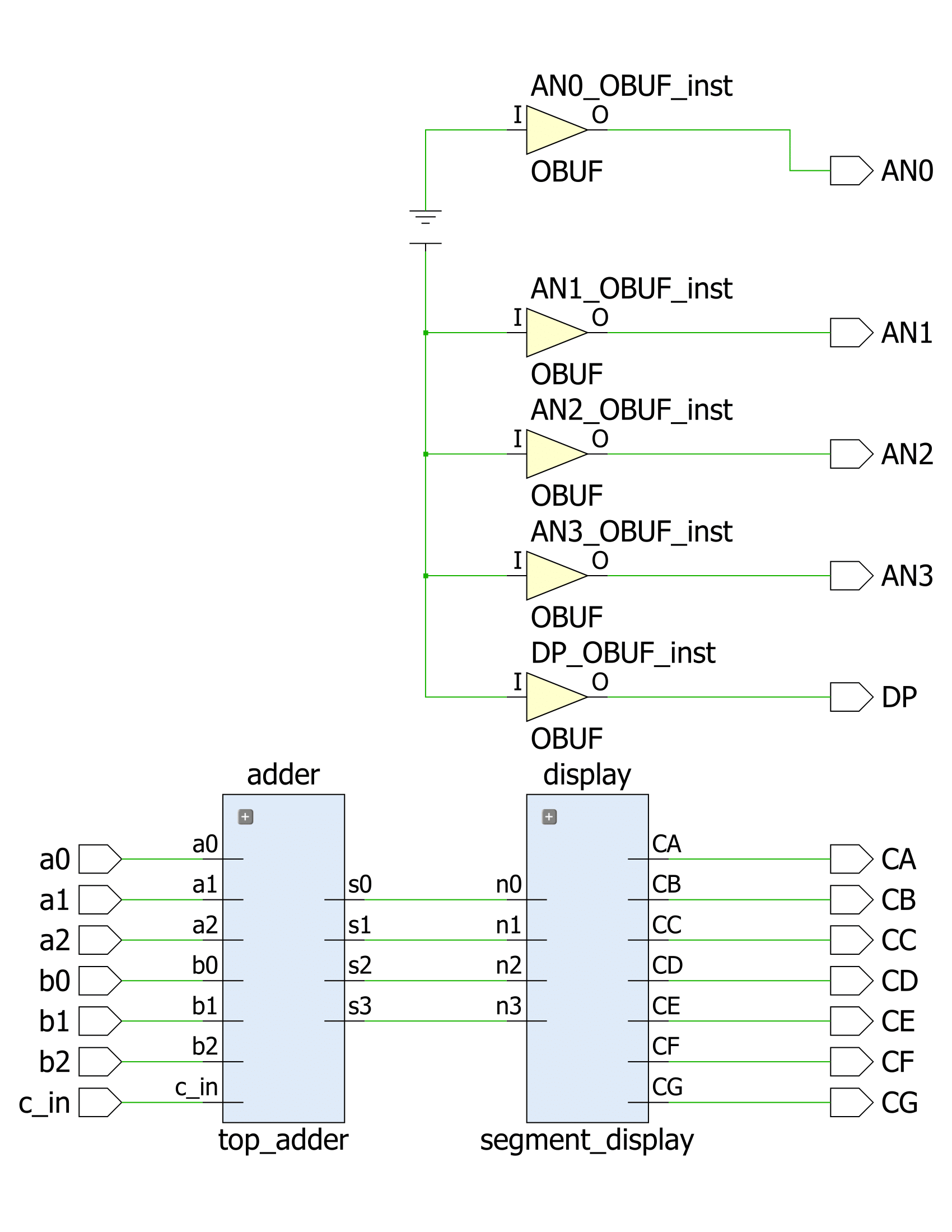
**Conclusion:**

In conclusion, we successfully built a functioning ripple-carry adder that added two 3-bit vectors and displayed the result onto a 7-segment display as a hex number. We learned how to use hierarchy of modules to repeat a function and keep our code more organized, and also how to use the simulator in order to test our code and display its result as waveforms. If I were doing this lab again, I would make better use of the simulator by testing my code as soon as I write it instead of testing it after creating multiple files with code that could be wrong.

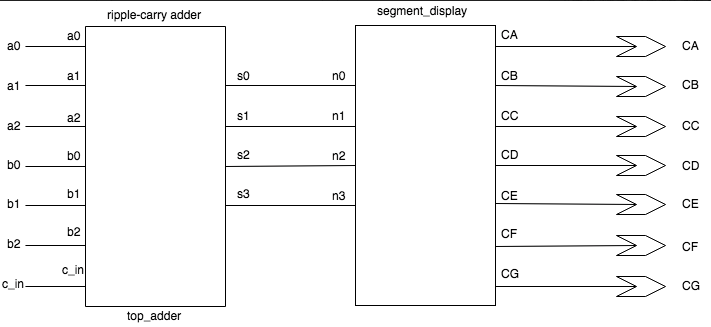
**Appendix**



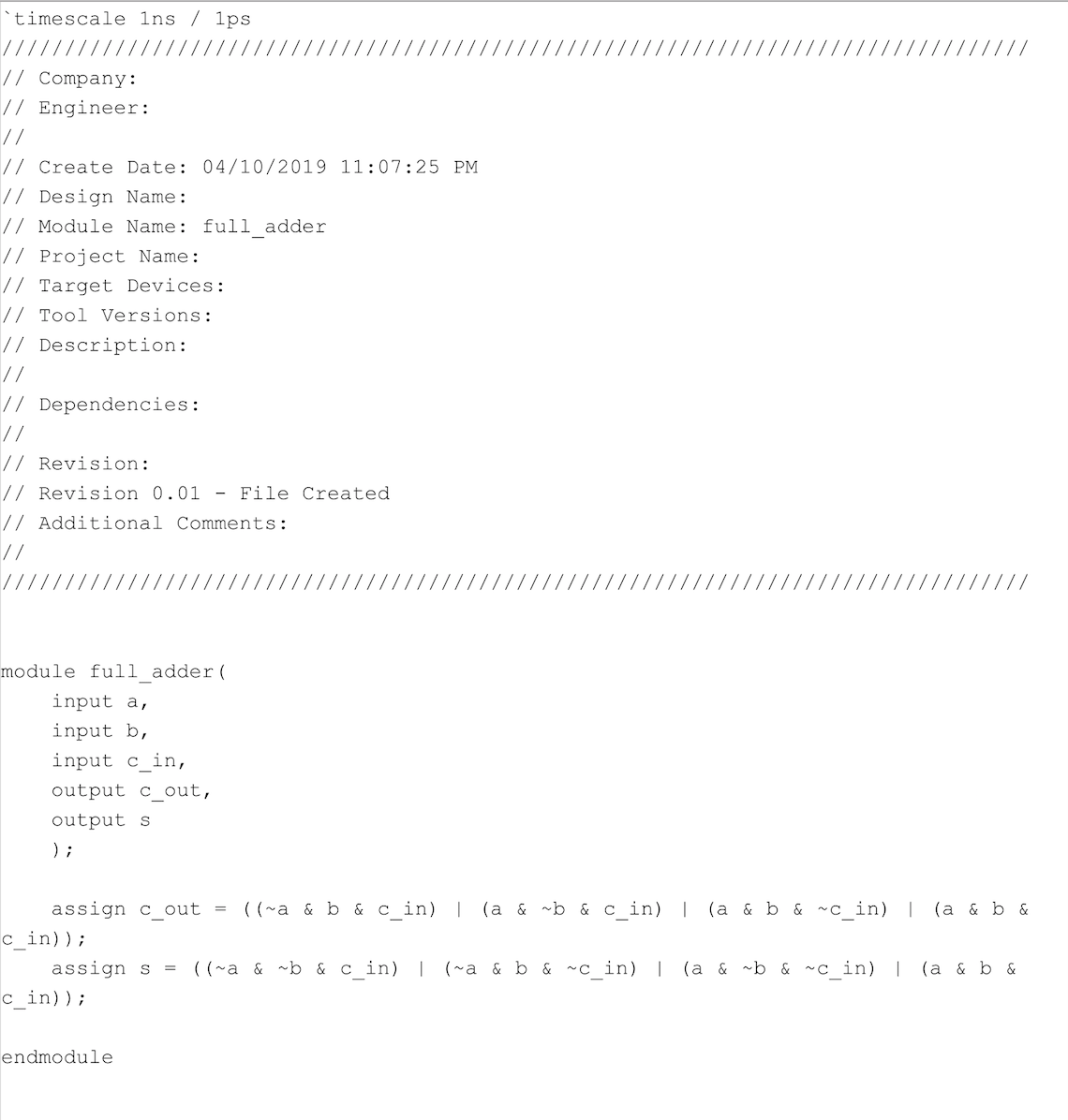
Ripple Carry Adder



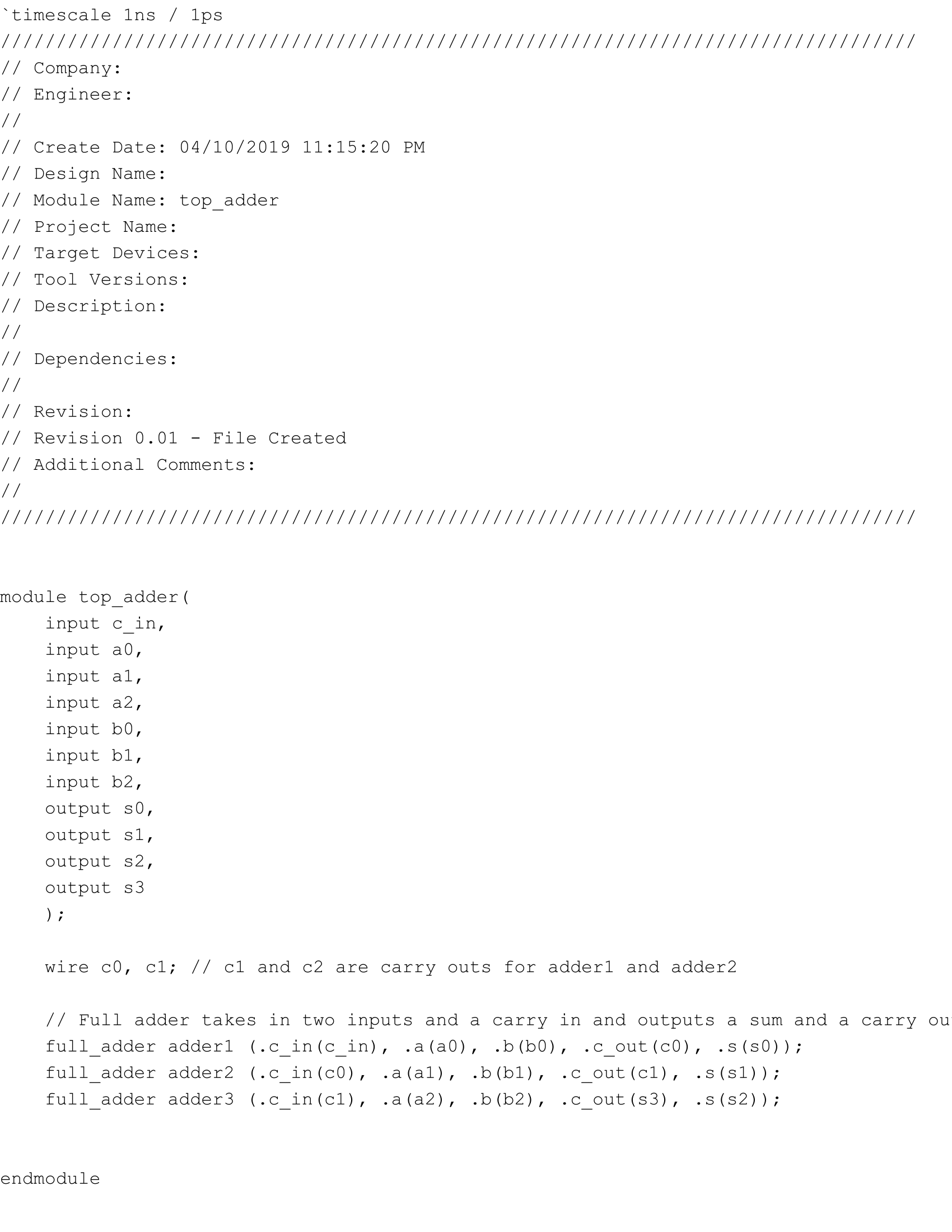
top\_mod schematic (Xilinx)



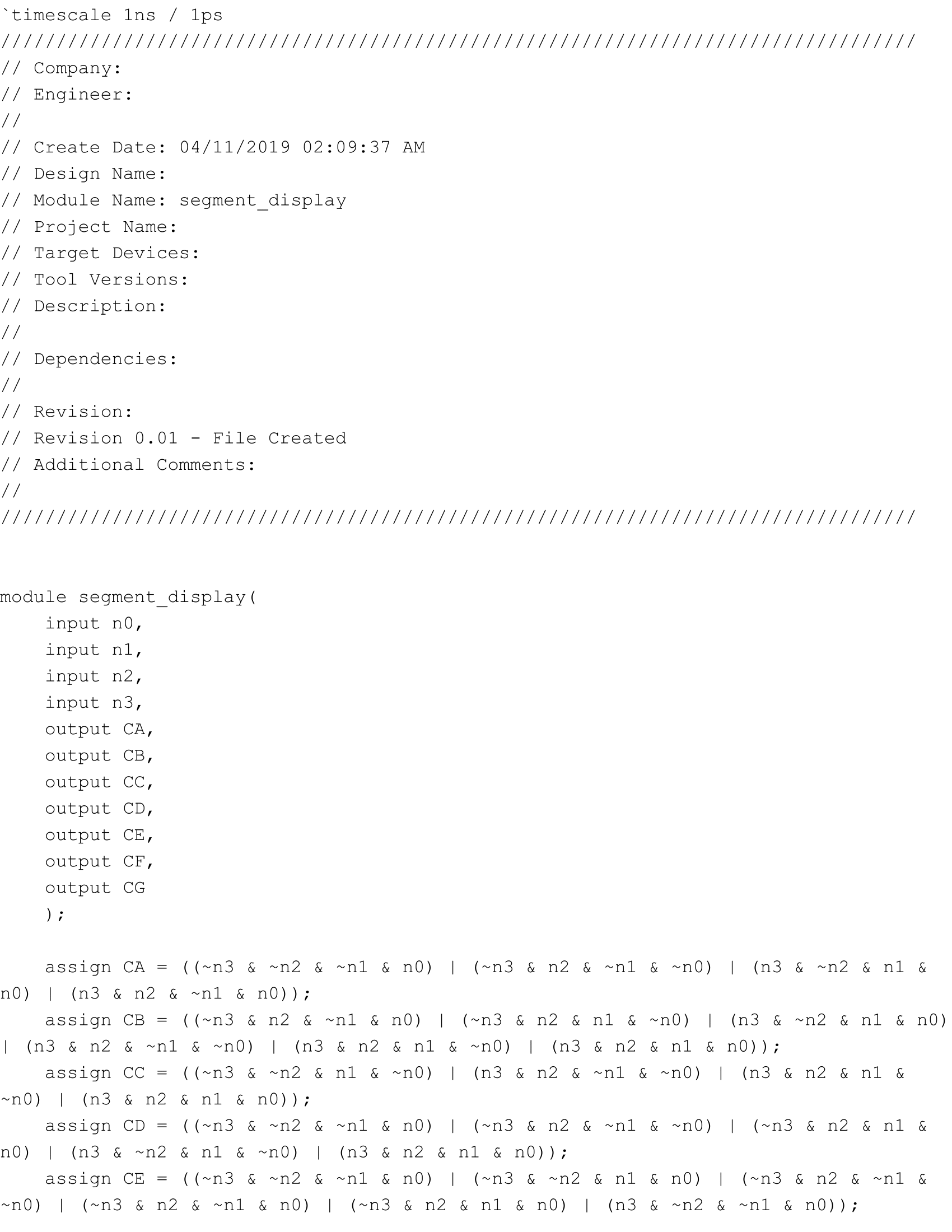
top\_mod schematic (Draw.io)



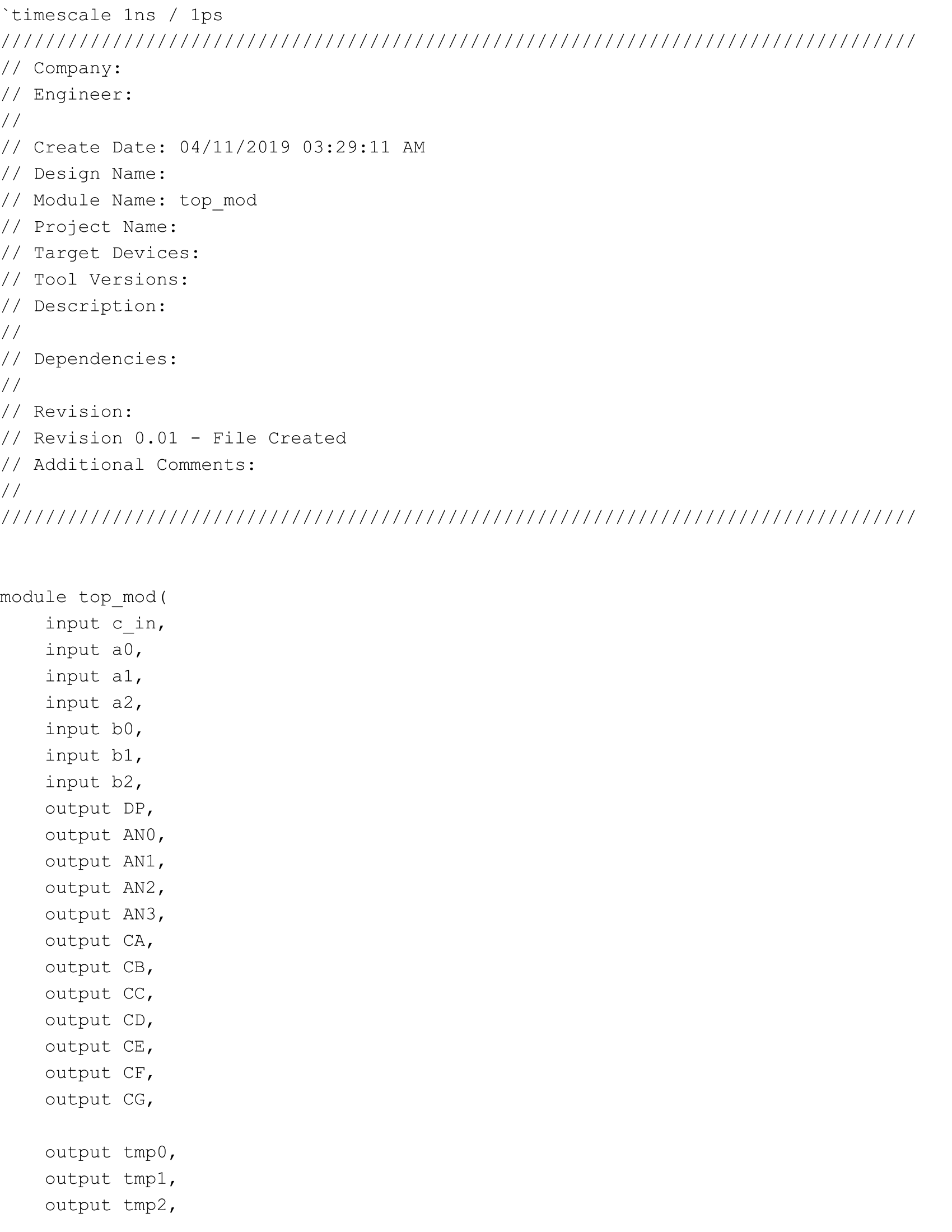
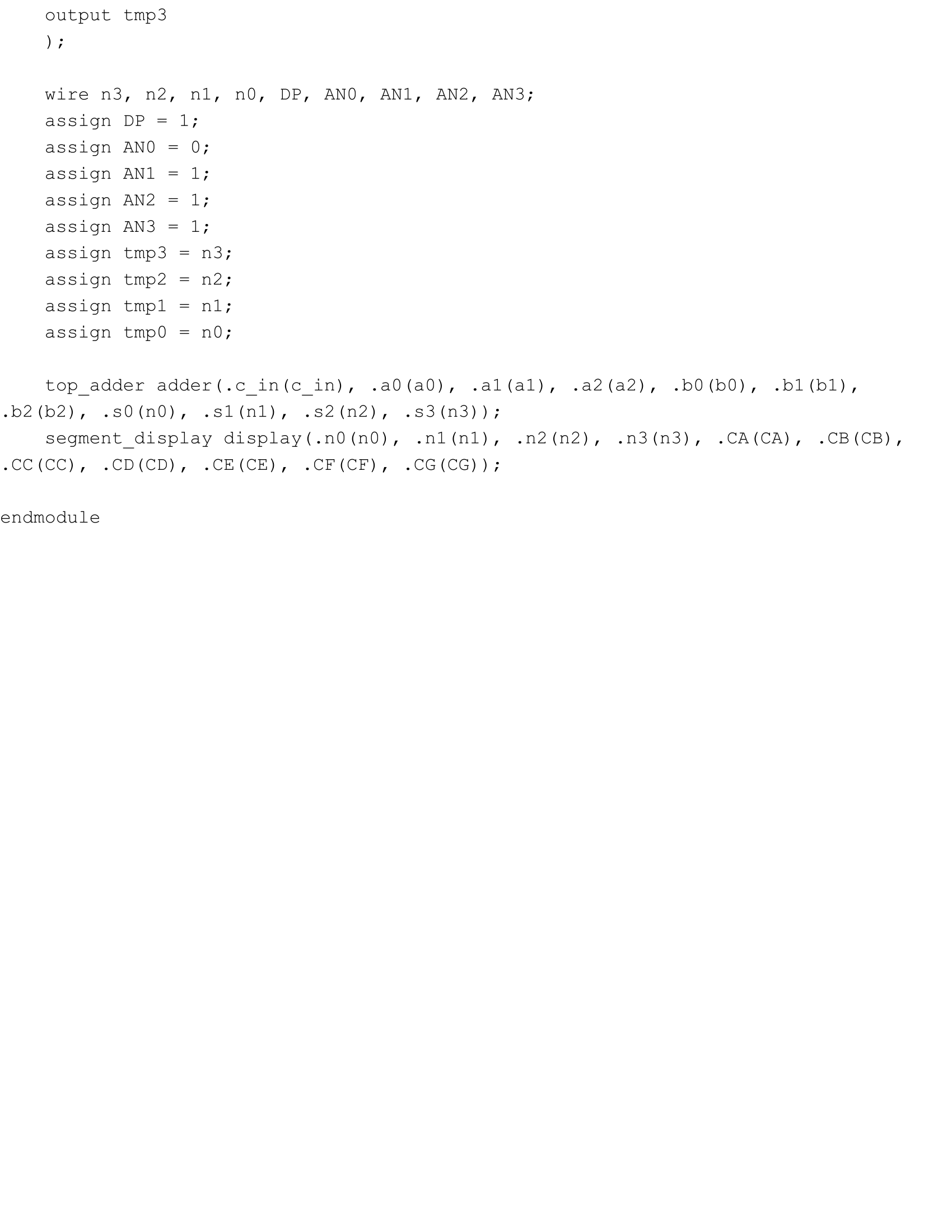
full\_adder.v



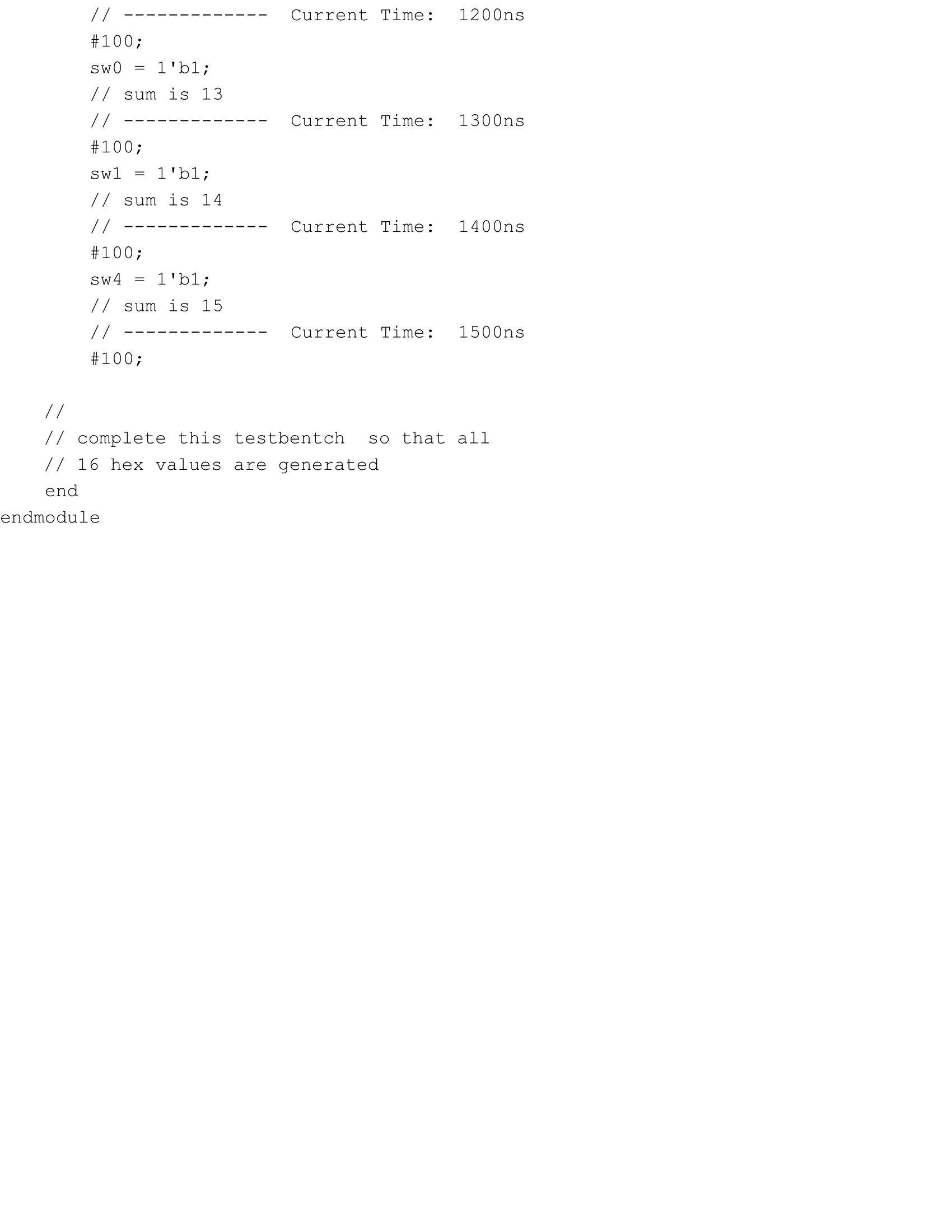
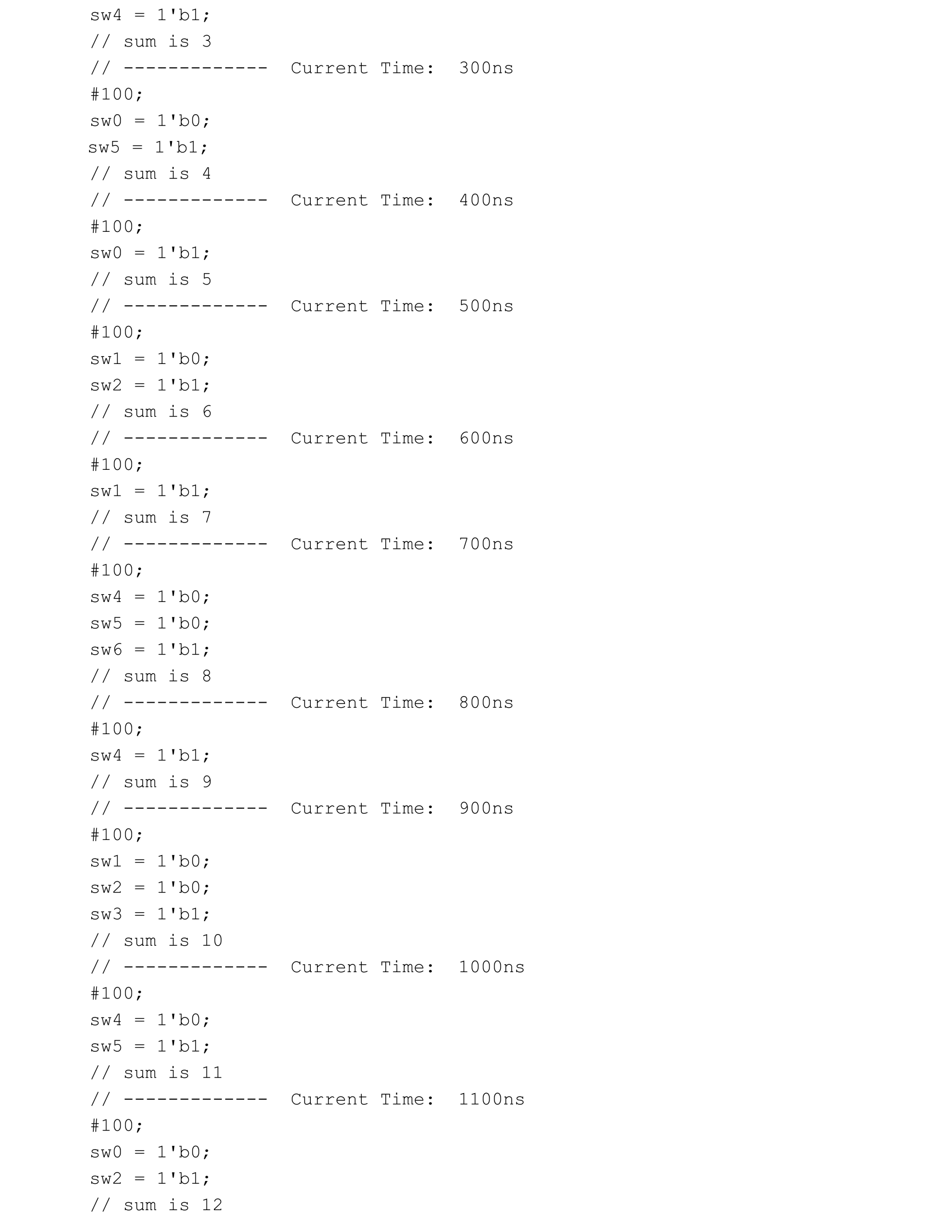
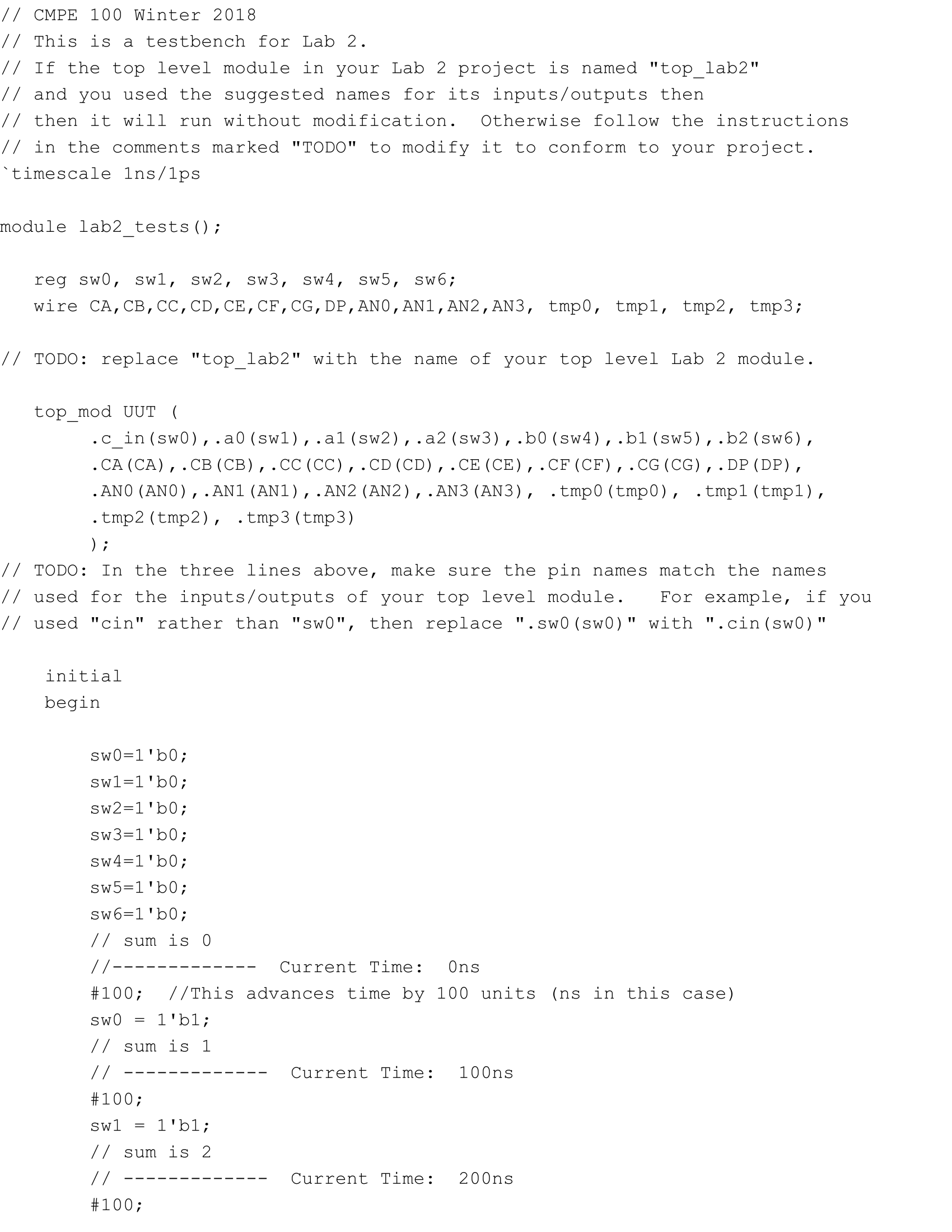
top\_adder.v

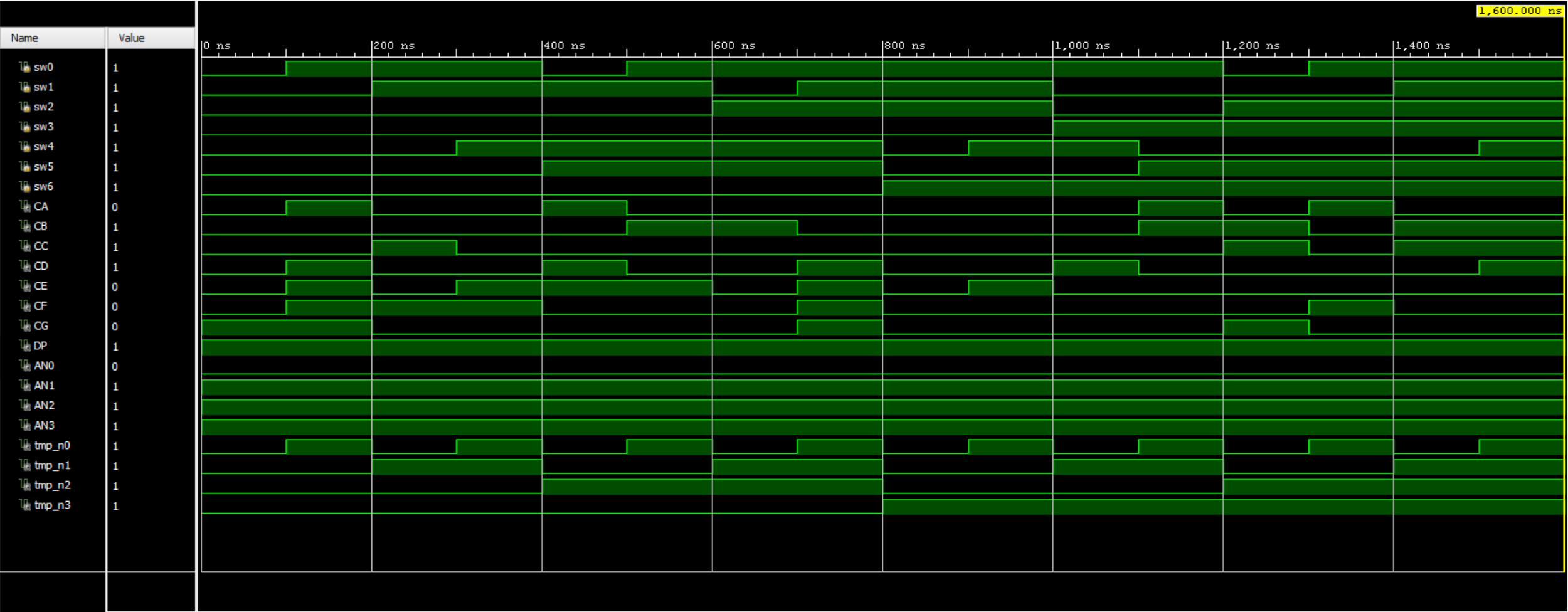
segment\_display.v

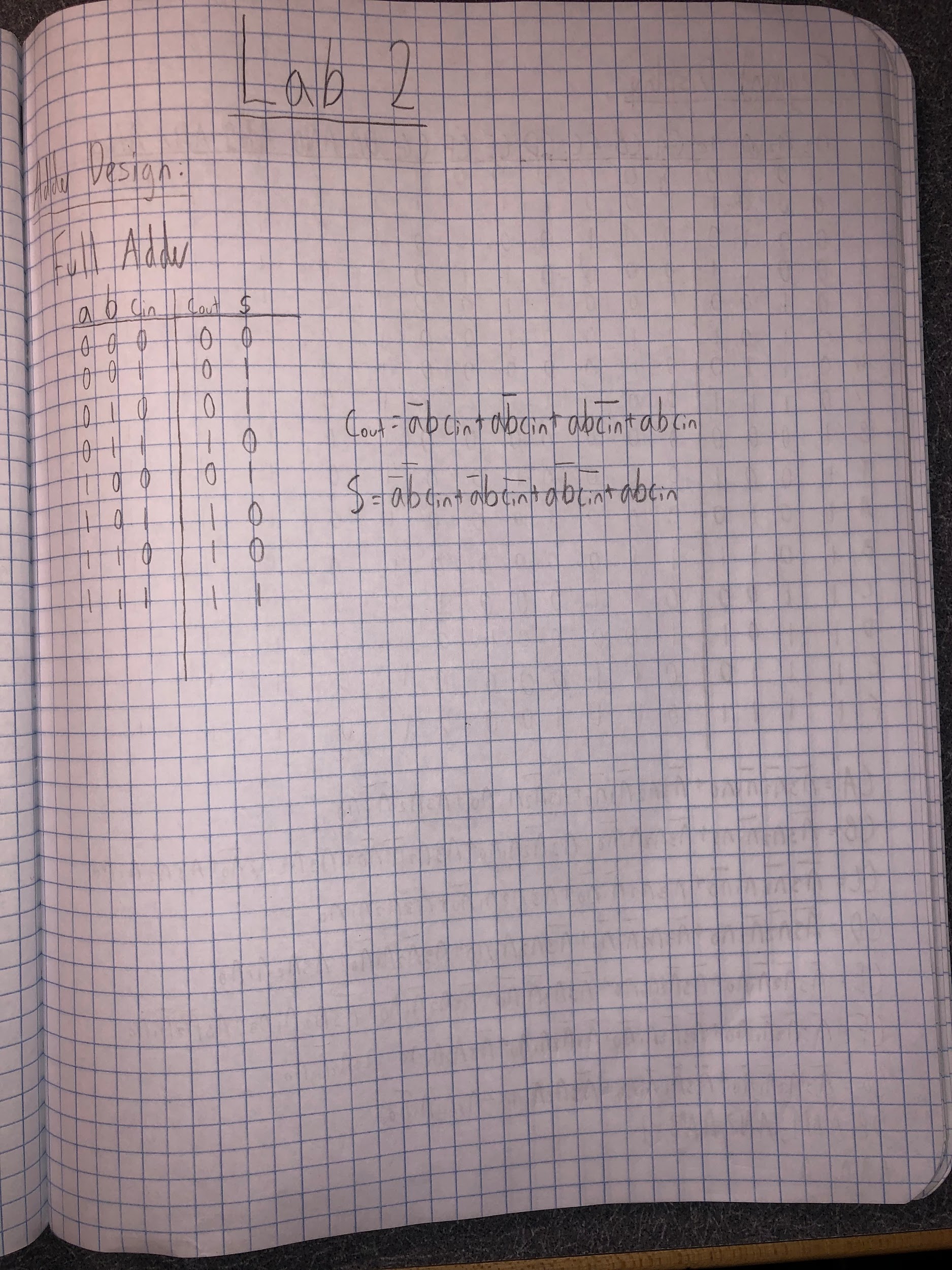
top\_mod.v



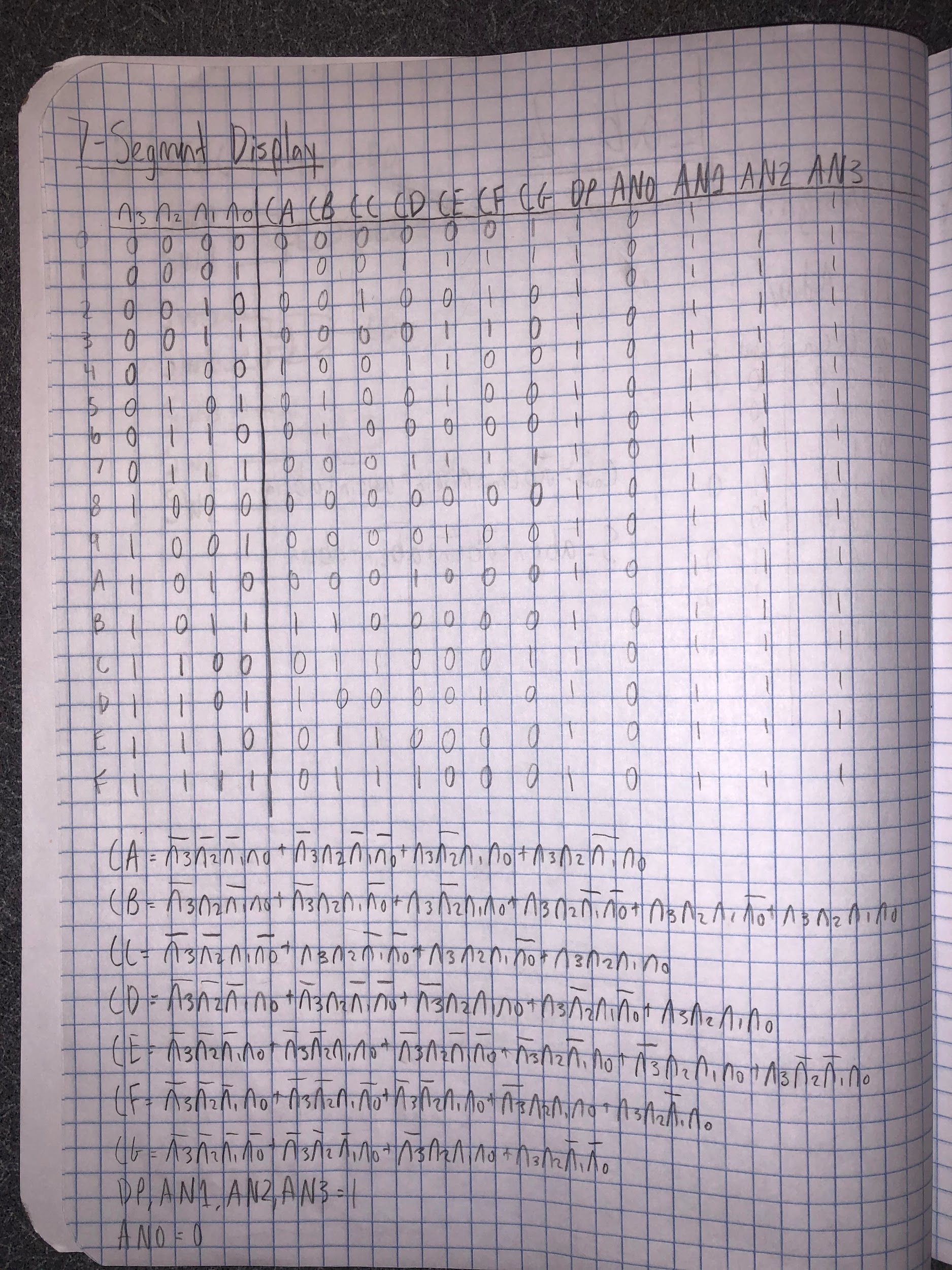
lab2\_tests.v



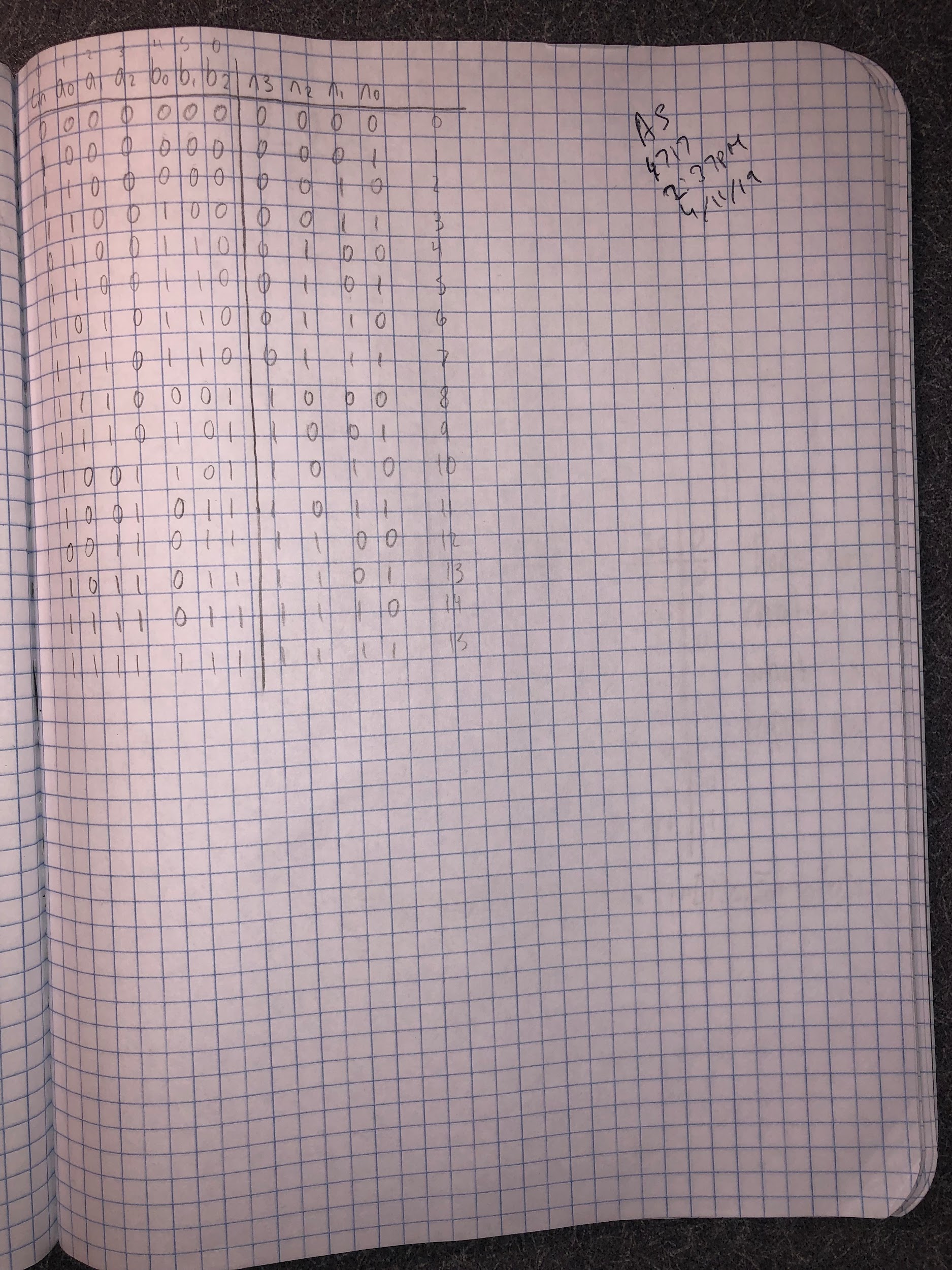
Waveform for top\_mod



Lab Notebook Page 1



Lab Notebook Page 2



Lab Notebook Page 3