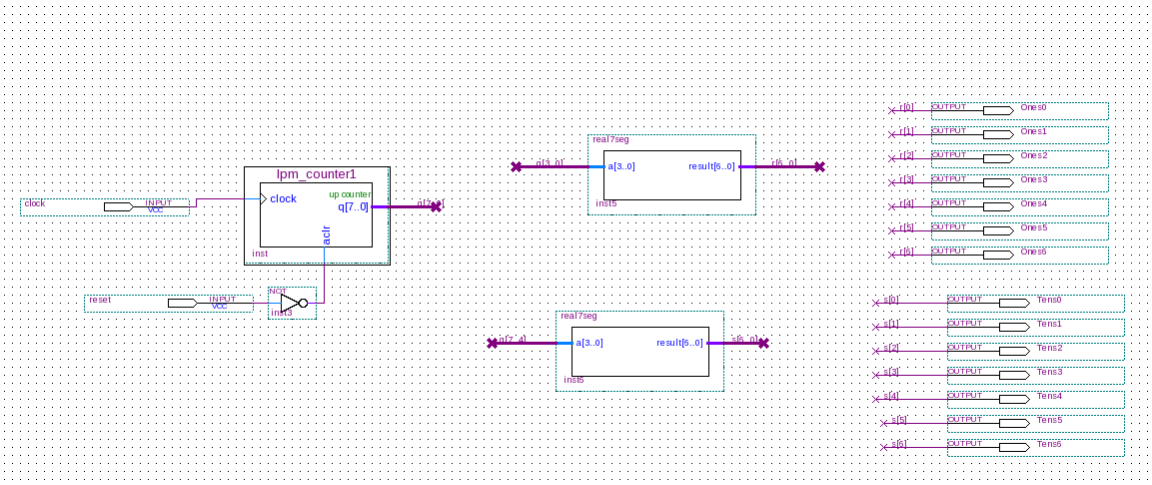
Zena's CS232 Project 2

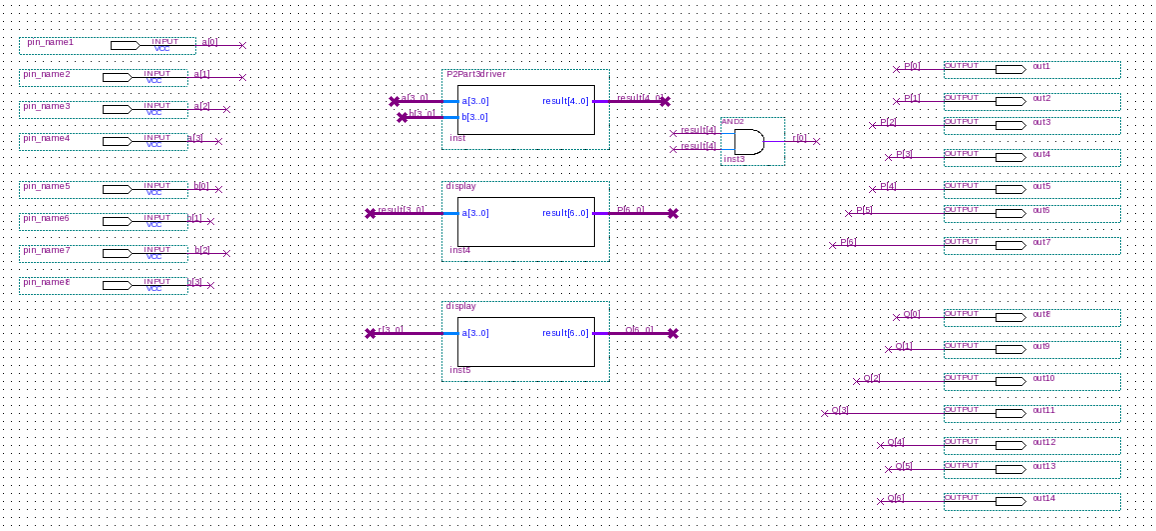
For Project 2, we built a hexadecimal (0-9 then A-F) counter, turned it into a two-digit hexadecimal counter, and then made a 4-bit number adder (which would result in a 5-bit number). The pins/buttons on the breadboard were used to control the number displayed on the seven-segment displays.

For Task 1, I created a circuit in VHDL that would take in 4 bits and use them to control the seven parts of the seven-segment display and create combinations of the segments to make numbers and letters. 3, for example, was made by having segments 0, 3, 4, 5, and 6 on. The decimal number 3 in 4-bit binary is 0011, so my VHDL file had segments 0, 3, 4, 5, and 6 equal 1 when the input was 0011. This was done for all 7 segments; each one was told which number to be on for. In order to change the number that is input, a counter is needed, which would count from 0000 (0 in hexadecimal) to 1111 (F in hexadecimal). To that end, I put a counter into a BDF (block diagram file), which counts up from 0000 to 11111, and which goes back to zero either after being pressed following reaching F or when the reset button is pressed at any time. The output from the counter goes through the symbol file version of the VHDL file (called "real7seg" here), then gets output to the 7 segments making up one of the 7-segment displays on the breadboard. Task 1's block diagram picture is not included, but follows closely to that of Task 2, pictured beneath its description.

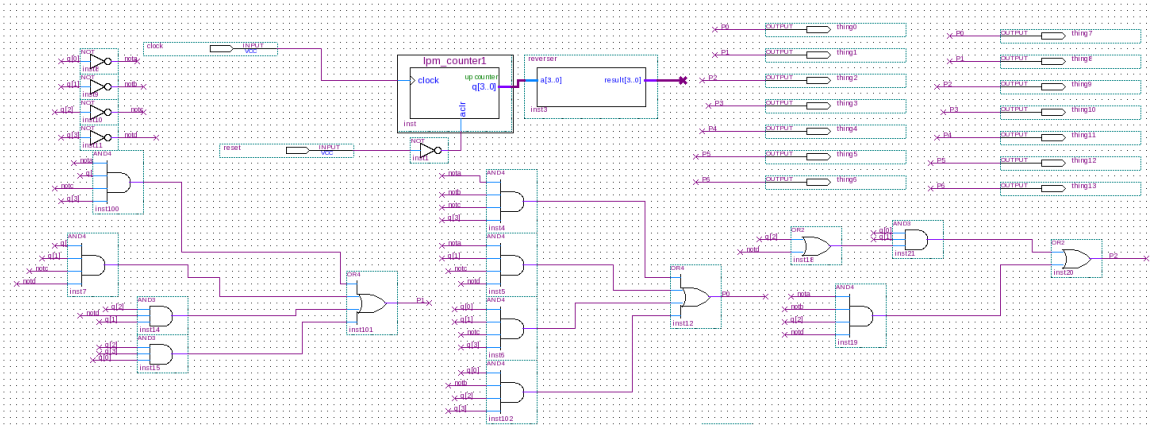
Task 2 was to make a counter that, like the one in Task 1, counts up from 0 to F, except this time, on two 7-segment displays, and in 8-bit binary. I only had to make a few changes from the files in Task 1. First of all, the counter for this goes up to 8 bits (0 to 7) instead of 4 (0 to 3). Next, I split the counter output into two halves-- one for the first seven-segment display (the ones' place) and another for the second (the tens' place). This was because the higher half of the counter is linked to the first set of 4, and after that, it naturally goes on to the first bit in the tens' place, then resets all the others below to zeroes again, meaning that linking the first four (left to right) bits to the tens' place output and the last 4 bits to the ones' place output would work. For instance, 0000 1111 would be 0F, then on the next count up, it would be 0001 0000, or 10. Therefore, the outputs displayed on the breadboard range from 00 to FF, or 0 to 255. The information on the segments for the provided number is input into the display, and this is done for both the first and second displays. An image of my block design is shown below:

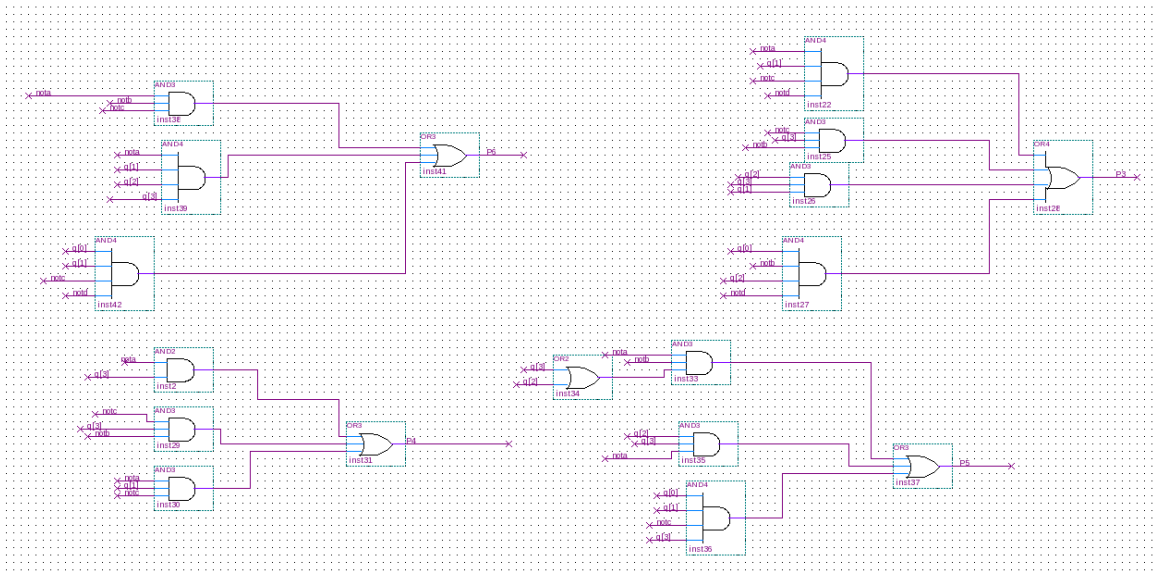


For Task 3, we had to build a circuit that adds two user-determined 4-bit binary numbers. Instead of a counter, the idea was to use 4 switches to control each bit of a 4-bit number. There would be 8 switches in total that would be used, the first four for the first addend and the next four for the second addend. The screen would display the sum in hexadecimal, on two seven-segment screens, much like the way the numbers showed up in Task 2. I started off having 8 inputs, connected to the adding driver I made in VHDL (called "P2Part3driver" here). The adder takes all of the inputs, adds them, and puts the first four (right to left) bits into the displaying driver. The most significant bit, result[4], goes through an AND gate (which was done to assign it to a new variable), then goes through the displaying driver. Both seven-segment displays then work together to show the result of adding any 4-bit binary numbers in hexadecimal. The image of my circuits is shown below:



I also did the first extension for this lab, which was to make the whole first circuit in a block diagram file, without using VHDL to control which inputs made which segments true, like before. This means that I had to find the boolean expression for each output using the original truth table I had used for determining when a segment should be true and by making Karnaugh maps. In this extension, I learned that I could make a shortcut for when a variable was not true; I put the original (ex: q[0]) through a not gate, then called it not[name of letter I associated it with] (ex: "nota"). After all 7 maps were done, I made a counter, which took two inputs, as usual: a clock and a reset button. I had trouble when trying the circuit on the breadboard, and noticed it started from 0000 then went to 1000, then 0100, then 1100, etc. which was in the *reverse* order from what it should have been. This led to me making a "reverser" VHDL file, which takes in 4 bits, and assigns to the output each bit of the input, but in the reverse order (so input(3) becomes output(0), for example). The output of the reverser is q[3..0], which is the correct order to count up in, then gets put into all of the gates, and goes through them to the 7 outputs. The picture of this circuit is below (please note that there are twice as many outputs here and that the output of the reverser is unlabeled).





In this project, I learned about how to represent circuits in VHDL and a lot about how counters work. I also became more familiar with hexadecimal notation and with solving problems through building drivers.