Eric Chuong

Rebecca Grob

ECE 425 Lab 2

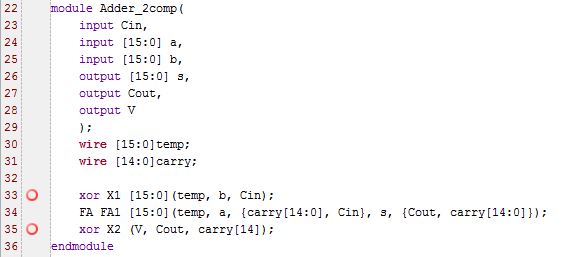
Introduction and Objective

The objective of this lab is to develop a structural model for an ALU. The ALU will be able to perform five different operations depending on an input, opcod. The five logical operations are OR, AND, ADD, SUBTRACT, and SLT (set if less than). We are also given several other inputs and outputs that must be included in the ALU.

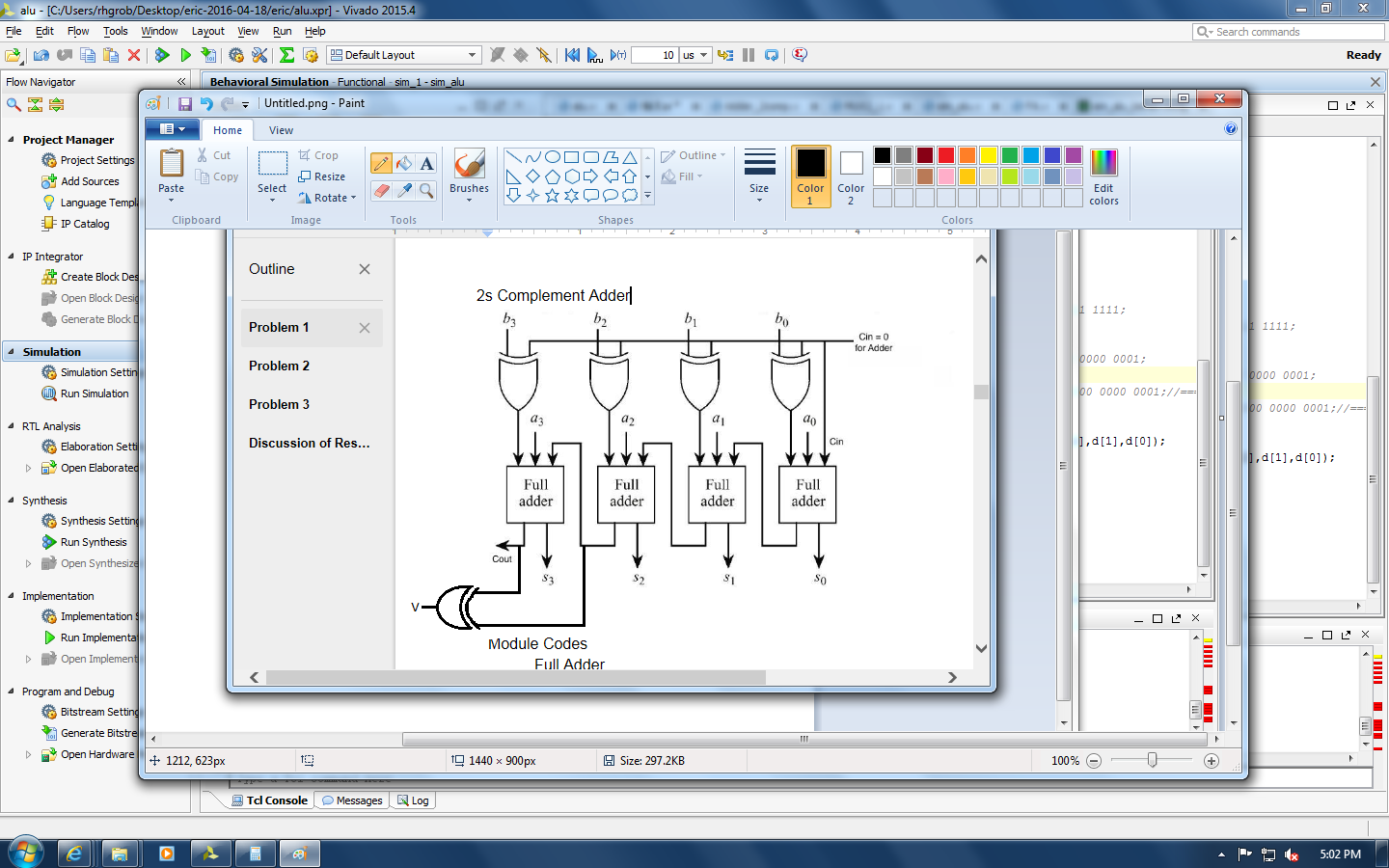
First, we show the codes that are required to construct the ALU.

**2’s complement adder**

**Code**

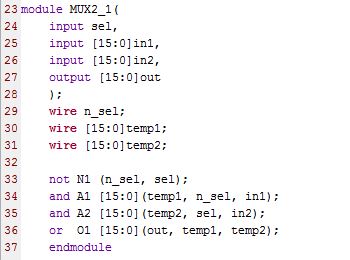


**Schematic**

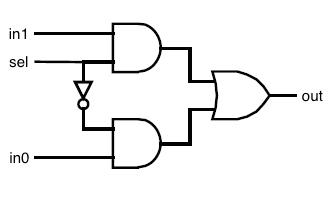


**16-bit 2-to-1 MUX**

**Code**

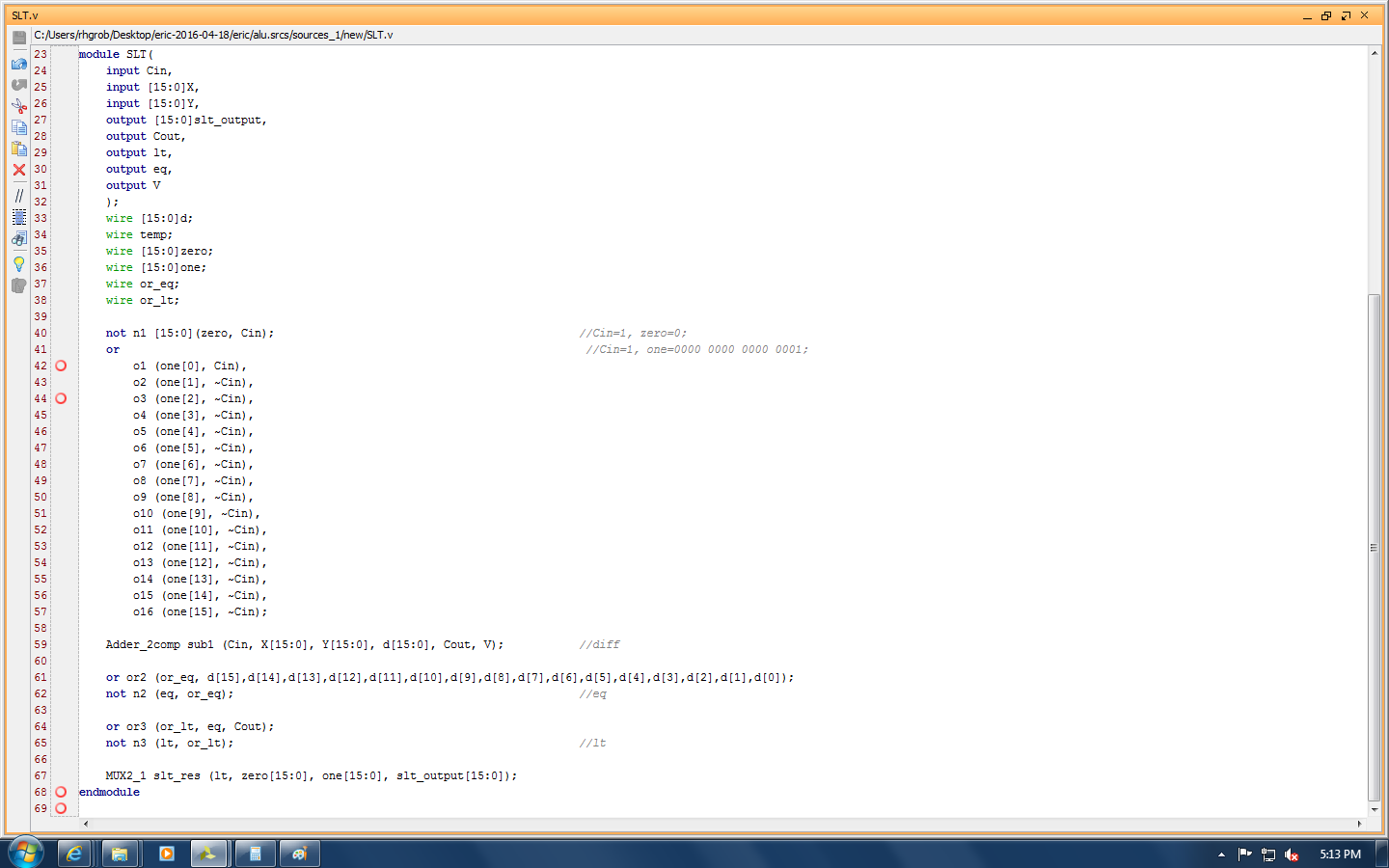


**Schematic**

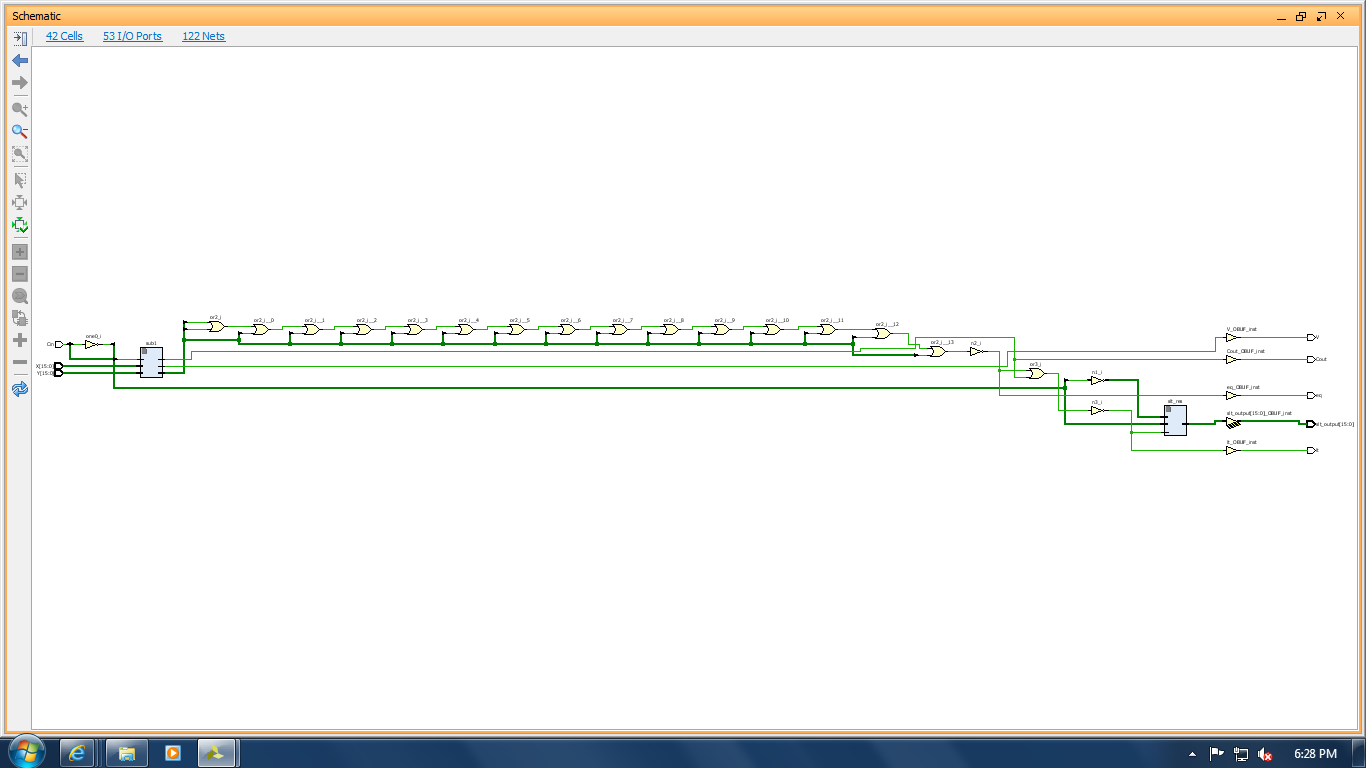


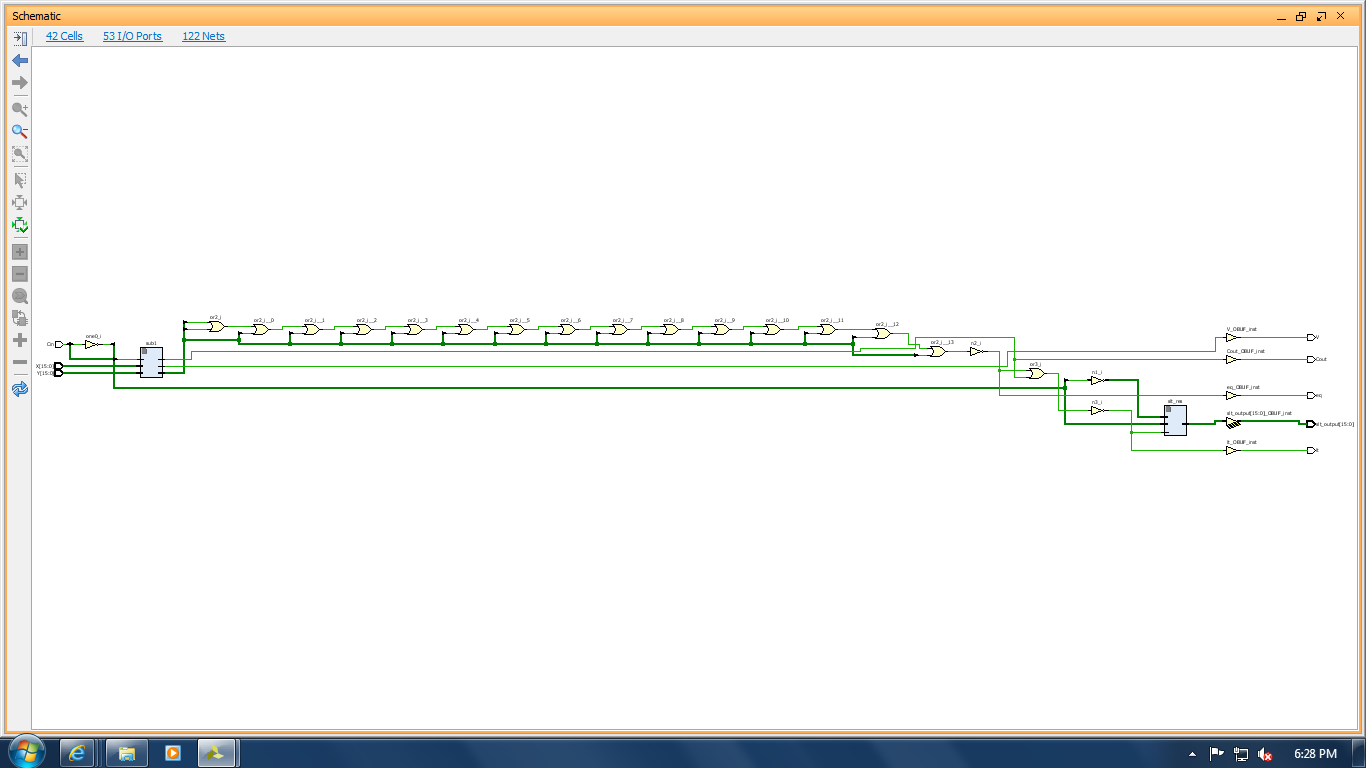
**SLT**

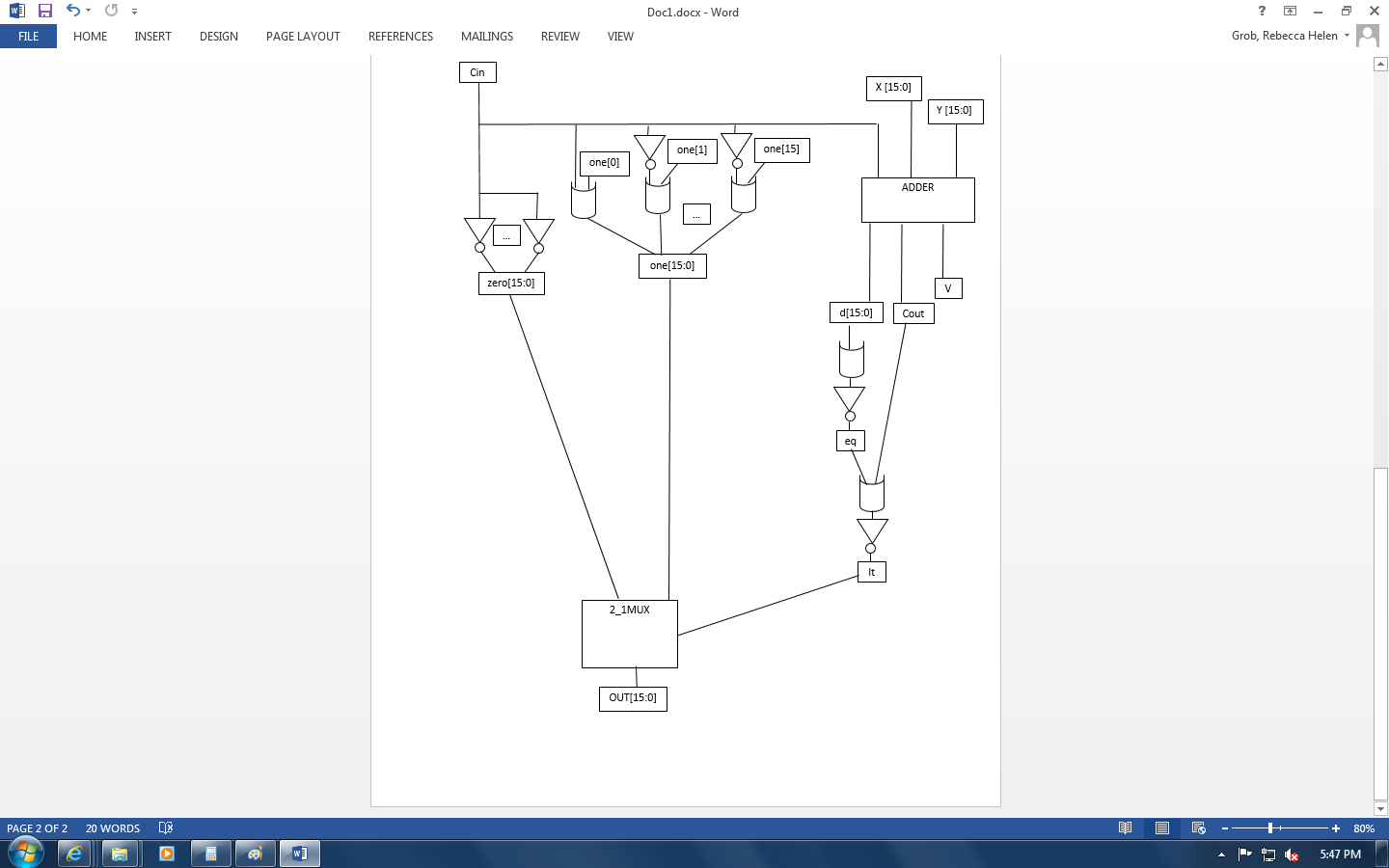
**Code**



**Schematic**

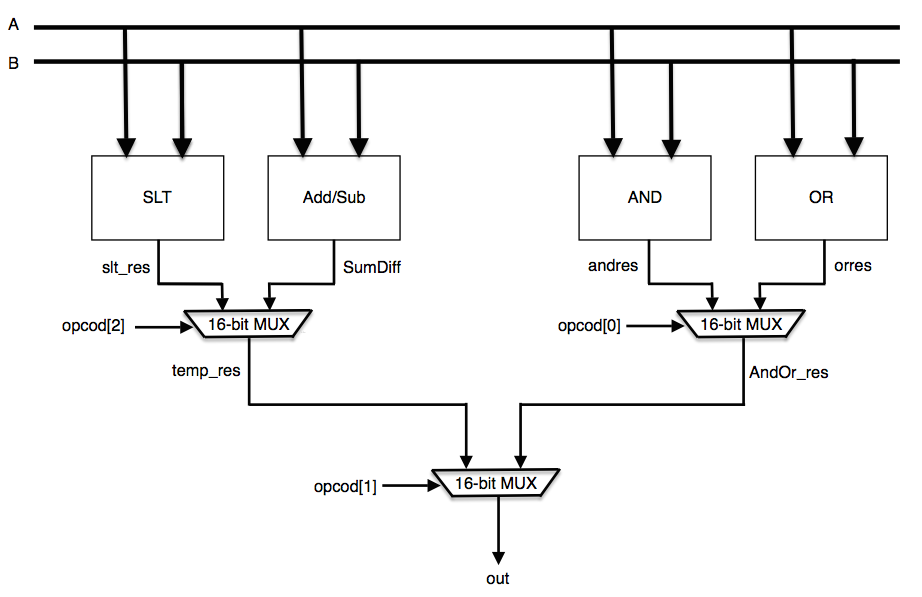


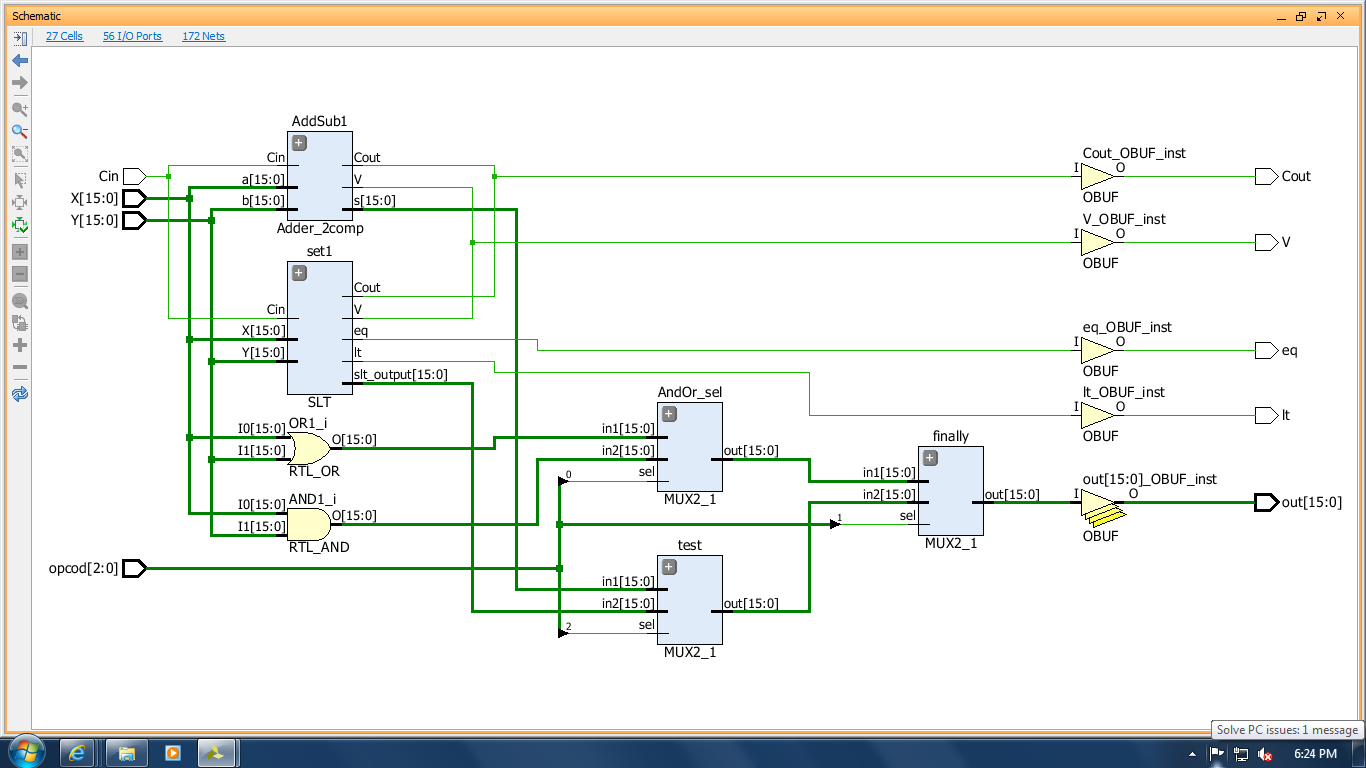




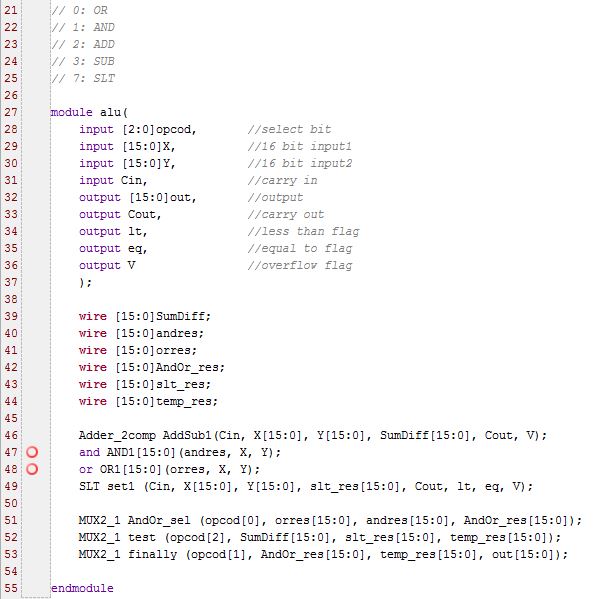
Now we have the actual ALU which implements all the previous modules

**ALU Schematic**

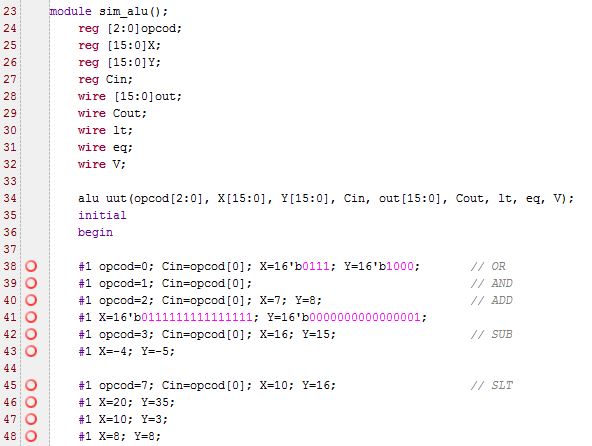


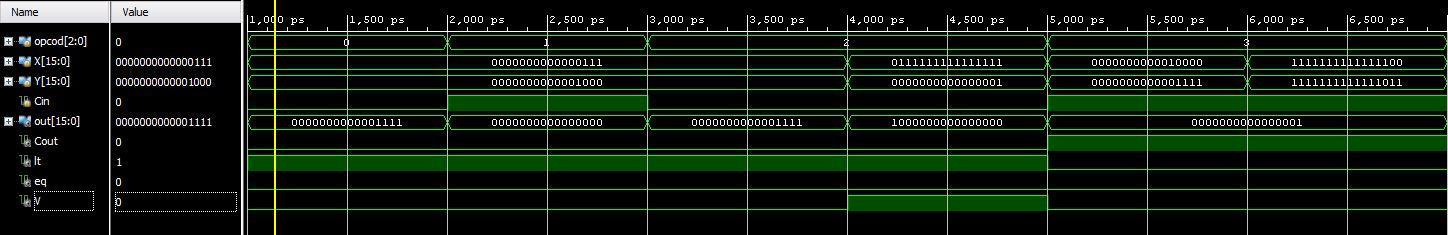


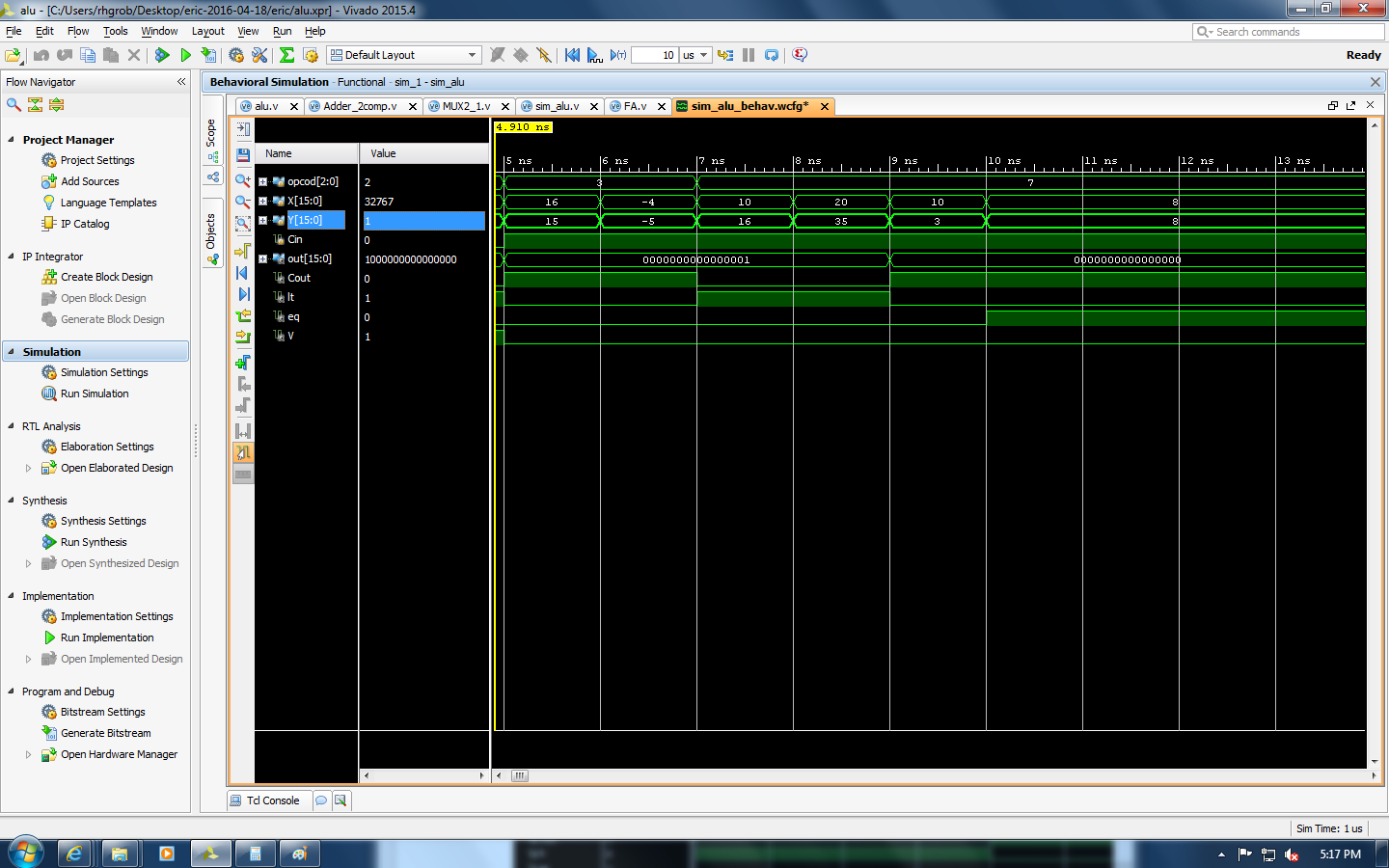
**ALU**

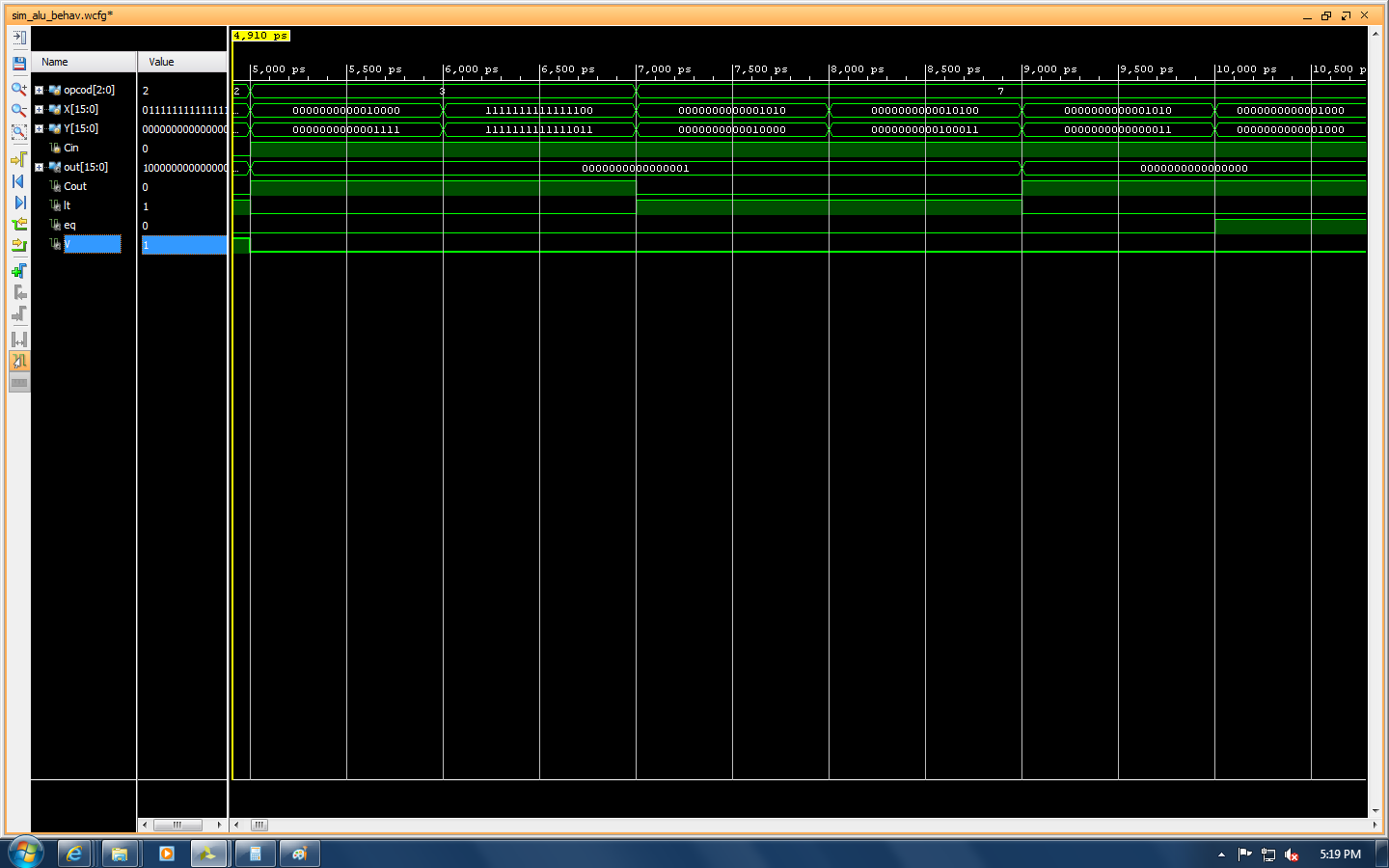


**ALU test bench**



**ALU simulation**





**Problem Analysis**

The MUX and the Adder/Subtractor were both created in the last lab and their respective codes were recycled. The new module would be the SLT and the ALU. For the SLT, the goal was to create a module that would accept two 16-bit numbers, X and Y, and if X was less than Y it would output all 1s. On the other hand, if X was greater than Y then we would generate all 0s. Along with this, we would output lt and eq flags if X is less than or equal to Y respectively. To perform this, we first created a wire that would be all 0s and one that would be all 1s. One of these would be our final output. Then we performed the operation Y-X using the Adder/Subtractor. We looked at each bit in the difference and if the resulting value was 0, we could conclude that X and Y are equal and thus we would set eq to 1. Then we performed an or with the Cout and eq, and took the inverse of it which would determine if X was less than Y. Finally, we called the MUX using lt as the select so if lt is 0, X is not less than Y and we would output all 0s. If lt is 1 then X is less than Y and we would output all 1s.

Now for the ALU, we simply performed each logical operation using wires to hold each result. We then used MUXs and opcod bits as the select to choose which operation we would output in the end. If opcod is 0, we output the result of the OR logic function. Similarly, if opcod is 1, we output AND. 2 we perform ADD, 3 is SUB, and lastly 7 is SLT.

For the test bench, we wanted to show at least one of each logical operation, focusing on changing the value of each input and output at least once. So we have opcod changing from 0 to 4 and then to 7. We show how the OR and AND work which are fairly straightforward. The ADD demonstrates simple addition as well as how V, the overflow flag, could be set to 1. The addition shows how the Adder/Subtractor words for both negative and positive numbers. For the SLT we showed when lt and eq are 1 as well as the output being all 0s or all 1s depending on the lt value.

**Discussion of Results**

Overall this lab was fairly simple because we were able to recycle code from the previous lab. The biggest challenge was to create the SLT operation because we haven’t worked with this very much in the past. In the end, everything worked out how we expected it to go and the ALU we designed is able to perform all the necessary functions. One thing we learned from this lab is that you can break down complex problems into more basic ones and each completed part acts as a building block to the final product.