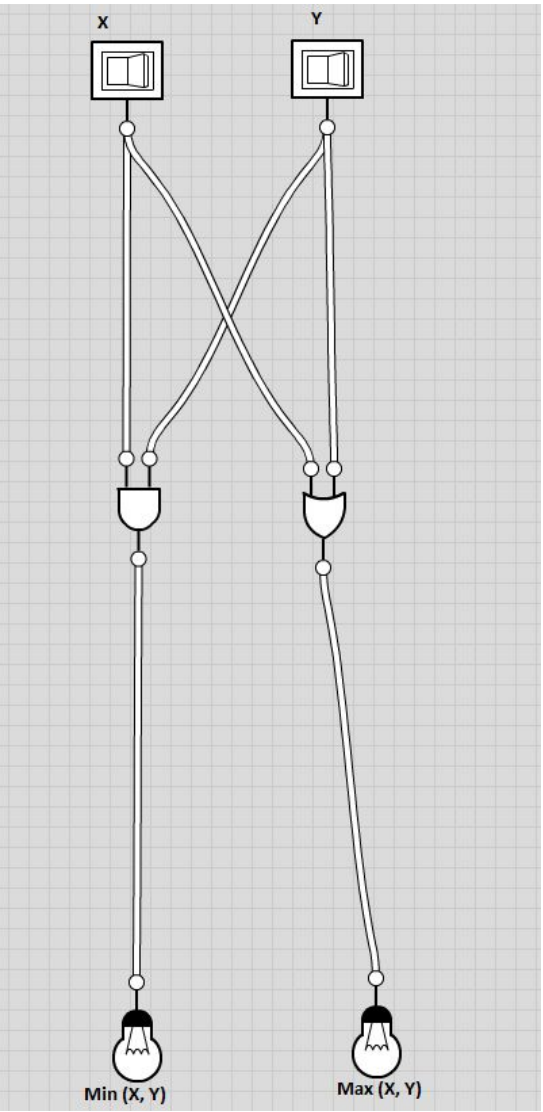
Kevin Gomes
Assignment due: 10/26/2017 at 11:55 PM

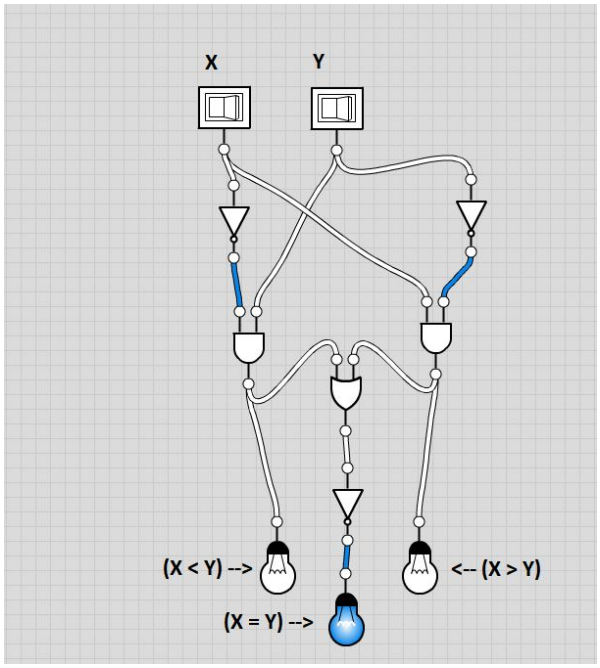
Part 1:
1.1: Design a 2-way 1-bit switch.



Fairly simple…
Part 1.2: Design a 2-way 2-bit switch.

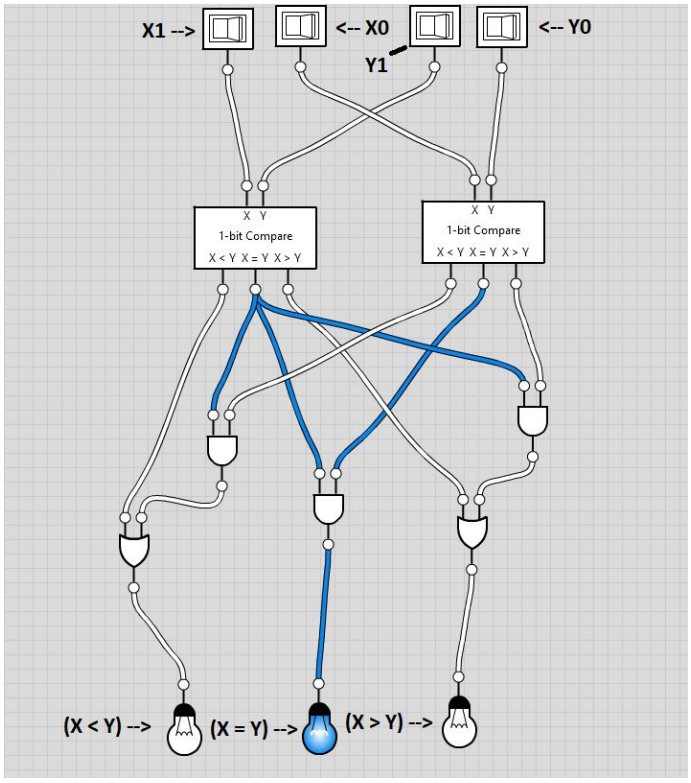
To start this, I had to make a 1-bit and 2-bit comparator to test for if a given X is greater than Y. We did a similar circuit on quiz #1, but I thought it would be good to extend this design with 3 outputs, just in case. It is as follows:

1-bit comparator:

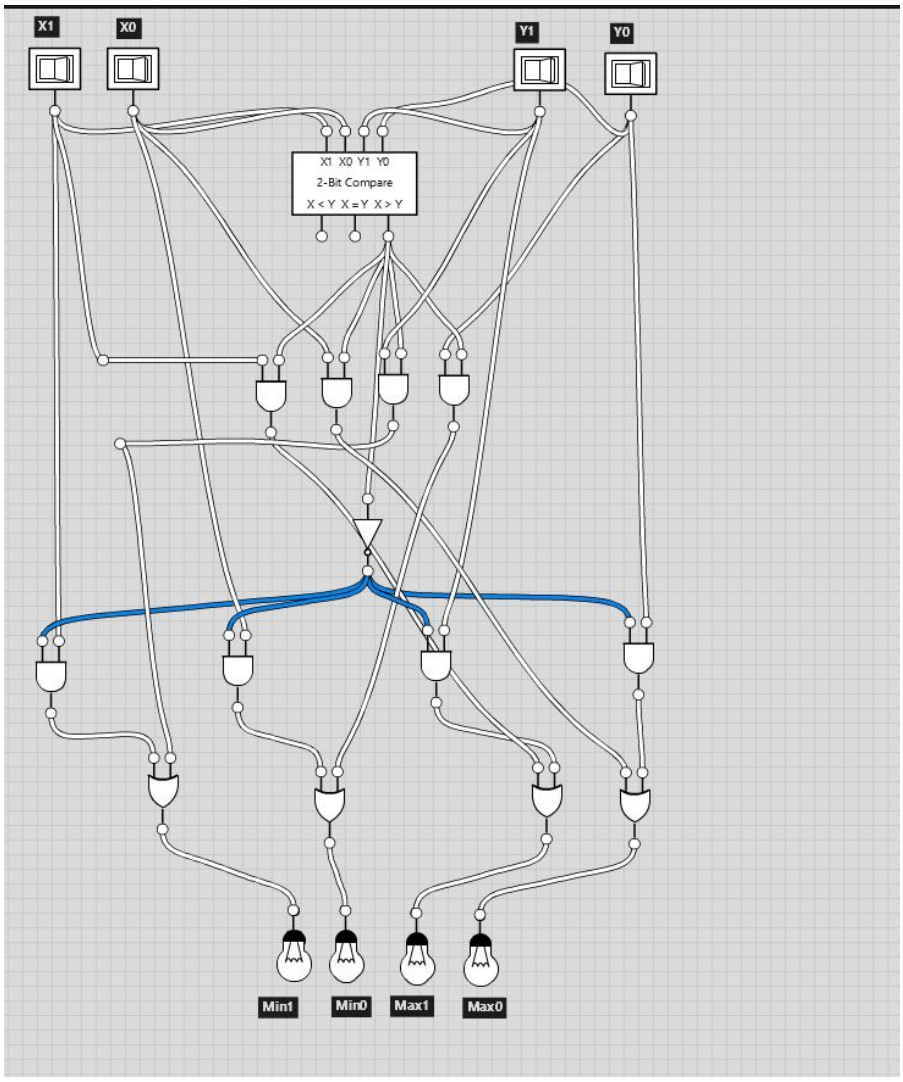


Next, we need a 2-bit version of this, as we are sorting 2-bit numbers. X1 is the most significant bit, X0 is the least.

2-bit comparator:



Finally, we use this in the final 2-bit shifter/sorter to test if X is greater than Y. If this is true, we swap the inputs. If it is not true, then X is less than or equal to Y. Therefore, we keep the inputs corresponding with the outputs.
2-way 2-bit shift:

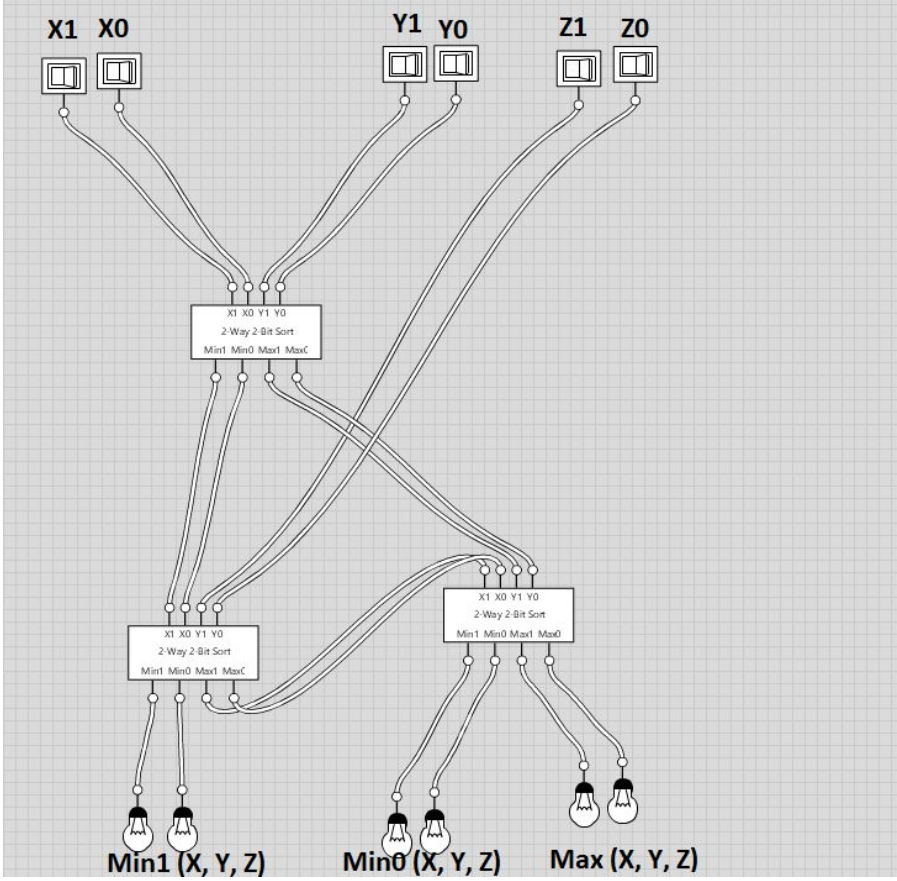


Part 1.3: For a K-Bit shift or sort, we would first need to increase the bits of our comparison. It seems foolish to be comparing the parts of a number instead of the whole. Therefore, we would need a k-bit comparator. Afterwards, we would need k ANDs, times 2. Or k*2 ANDS. One set is for if X > Y, the other set is if X <= Y. We would still use 1 NOT, and k ORs at the end. We can thus generalize this for any k-bit. This could be made more modular I feel however, using a multiplexer or 2 and the 1-bit sort (as it is customary to use k-1 bit sort to do a k-bit sort), but that is not how this is implemented.

Part 2:

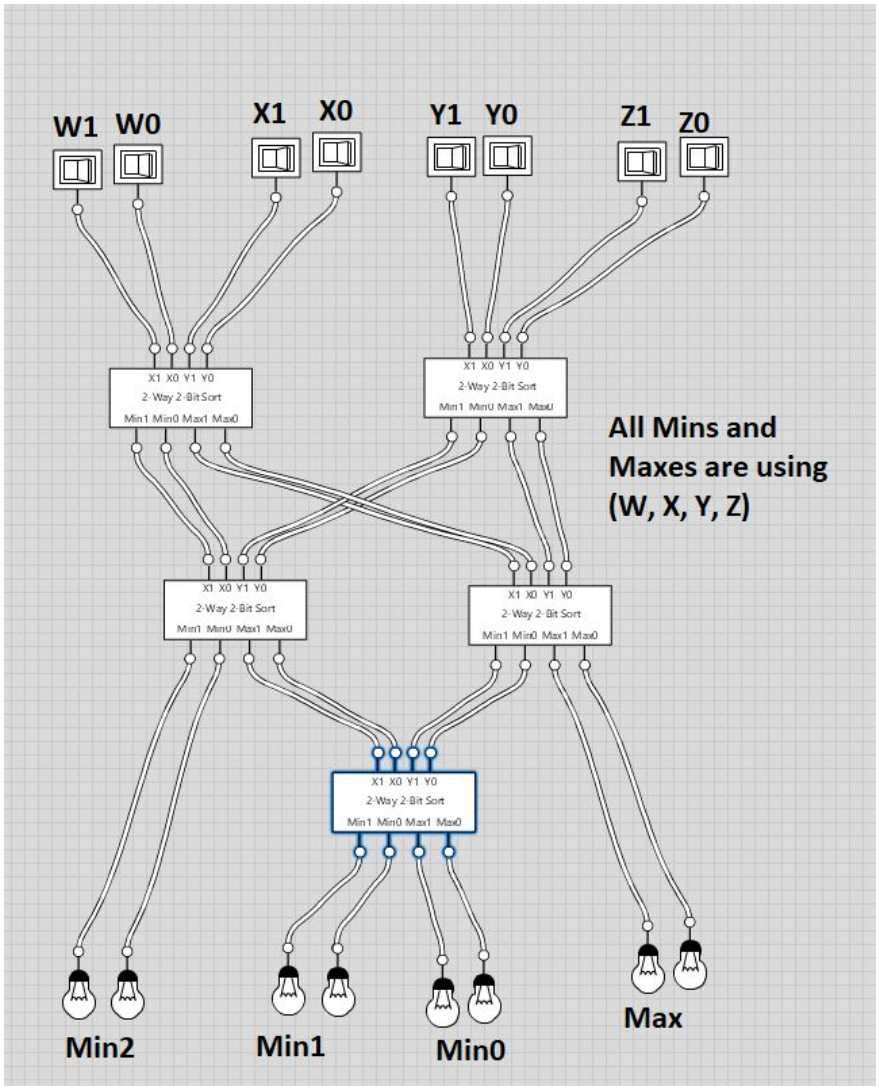
Part 2.1: For this part, we employ modular design and use many 2-way 2-bit sorts to sort 3 different 2-bit numbers. Of course, because we could employ this with k-bit, this is just shown as a demonstration, but in reality we would insert 2 of our k-bit numbers into k-bit sorters, then put one of the k-bit numbers as the Y of our second sorter, and the X would be the min of the first sorter. This determines the lowest number. Next we compare the max of the second with the max of the first, and whichever wins the max is the largest number. The remaining number is thus the middle.

3-way 2-bit sort (could be k-bit by simply changing to 3 k-bit sorts and 3 k-bit inputs)

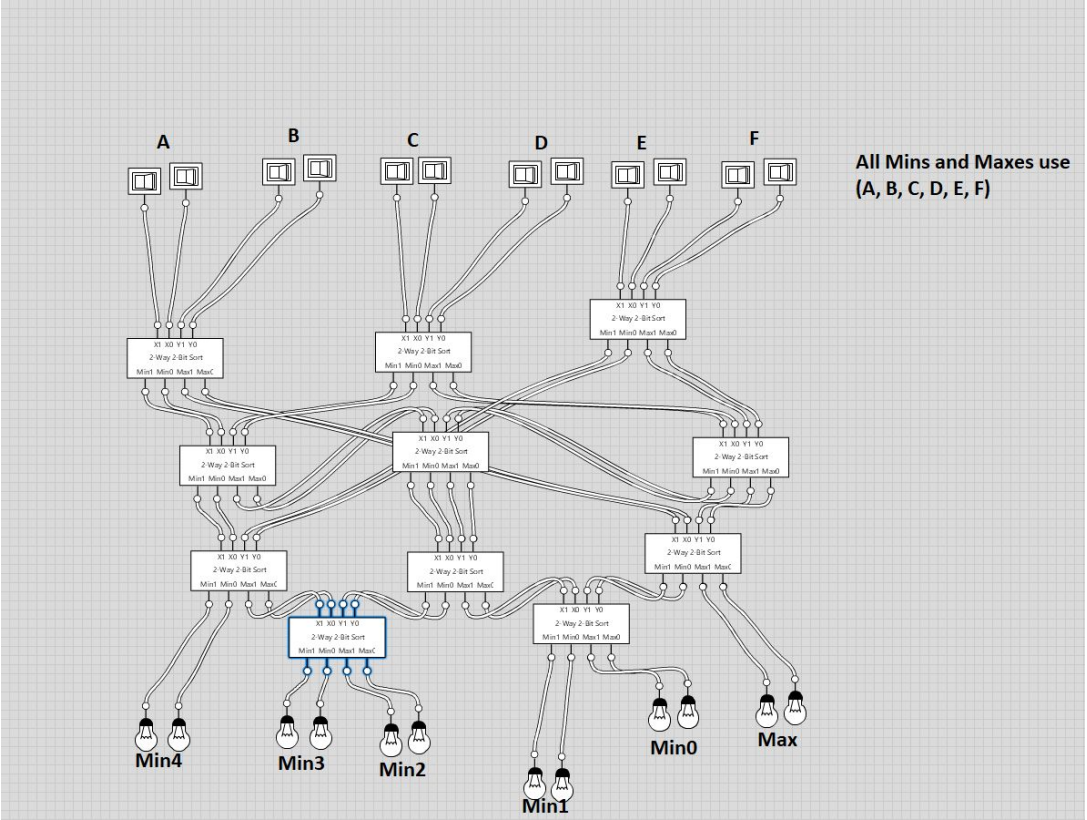


Part 2.2: Same concept as n = 3, but more boxes. Again, can use k-bit numbers by changing the inputs to k-bit and the sorters to k-bit. Also note that this can be more modular by using the n-1 way k-bit sorter, shortening the amount of circuits needed. This was not implemented in this way however. We also change slightly by sorting 2 numbers at a time at the beginning, and can do this whenever we have an even amount of inputs.

4-way 2-bit sort (could be k-bit by simply changing to 3 k-bit sorts and 3 k-bit inputs):



I've also included a 6-way 2-bit sorter to show how this can keep going for supporting my answer to part 2.3, found below:



Part 2.3: Clearly the question this assignment has been leading to is how to sort any amount of numbers, of which can be any amount of bits and thus any magnitude? Were I to implement this in the most correct way, our answer would be to use the n-1 way sorter of k-bits, and use that to sort n-way with k-bits. However, the way we implemented, we would generalize this to simply keep increasing the amount of k-bit sorters for each additional way of sorting (refer to the 6 way sorter above). It would have better to implement this by making ever n-way sorter a circuit, and using it in the n+1 way sorter. Alas, hindsight is 20/20.