Requirements:

- [x] ~~The design must be hierarchical (i.e., using blocks which you can reuse)~~
- [x] ~~One of the modules must be a 4-bit ripple carry adder~~
- [x] ~~You must use Structural Verilog (gate-level specification) for the RCAs. You may use Behavioral Verilog for the MUXes~~
- [x] ~~You should use Behavioral Verilog for the verification logic. This will be presented in discussion sections.~~
- [x] ~~Base the 64-bit adders on 1-bit full adders made up of ANDs, ORs, and NOTs. In particular, if you use XORs or half adders, then you must create a module first and build them up from ANDs, ORs, and NOTs.~~
- [x] ~~For timing, set ANDs and Ors to count as one gate delay and NOTs to zero gate delays. The behavioral MUXes should also have zero gate delays.~~
- [x] ~~Create two versions of all of the adders:~~
    - [x] ~~Using standard logic gates.~~
    - [x] ~~Using the "timed" gates to be provided.~~
- [x] ~~Your timing diagram must include representative examples. This is relatively easy in this lab, but becomes more important later in the semester.~~
    - [x] ~~a. Force an overall carry out~~
    - [x] ~~b. Random large A&B input (w/ and w/o carry)~~
    - [x] ~~c. Random small A&B input (w/ and w/o carry)~~
    - [x] ~~d. Random combinations~~

Turn in:

- [x] ~~.v files~~
- [x] ~~timing results description~~
- [x] ~~waveforms from testbench~~
- [x] ~~question answers~~

Questions:

What are the times of your adders in gate delays?

The timing varies depending on the size of the input. For small inputs, both the CSA and RCA are similar. As the input size increases, the RCA begins to struggle to keep up, and I wasn't able to extend the testbench window enough to keep up. The CSA handles the larger results much better, and does not increase in runtime at the same rate as the RCA.

Are they what you expect? Why or why not?

Yes, the RCA takes much longer to process larger inputs, since these inputs need to "ripple" throughout the entire structure. The CSA is able to do many more comparisons in parallel, so it is significantly faster as the size of the inputs increases. It's worth noting that the theoretical delay for the RCA is 128 gate delays, which is derived from 16 copies of a 4 bit RCA. For the CSA, this delay is cut in half to 64 gate delays, as it runs each half of the computation in parallel.

Timing Result Description:

The results with no delay are fairly similar between both the RCA and CSA. When running the RCA with large enough delays in the testbench to let it catch up to the new input values, it clearly struggles as the inputs increase. The CSA has a similar run speed to the RCA at low input values, but runs significantly faster as input values increase. The screenshots are as follows:

- *CSA_0delay.png*
    - *The waveform of the carry select adder with no gate delays*
- *CSA_1delay*
    - *The waveform of the carry select adder with 1 gate delay, with testbench delays of 10*
- *CSA_1delay_bigtb*
    - *The waveform of the carry select adder with 1 gate delay, with testbench delays of 100*
- *CSA_1delay_behaviorcheck*
    - *The waveform of the carry select adder with 1 gate delay, with a behavioral check to ensure the correct answer*
- *RCA_0delay*
    - *The waveform of the ripple-carry-adder with no gate delays*
- *RCA_1delay*
    - *The waveform of the ripple-carry-adder with 1 gate delay, with a testbench delay of 10*
- *RCA_1delay_bigtb*
    - *The waveform of the ripple-carry-adder with a gate delay of 1, with a testbench delay of 100*
- *RCA_1delay_behaviorcheck*
    - *The waveform of the ripple carry adder with 1 gate delay, with a behavioral check to ensure the correct answer*

.v File Descriptions:

- *CSA_64bit.v*
    - *Carry select adder top module (1 gate delays)*
- *RCA_64bit.v*
    - *Ripple carry adder top module (1 gate delays)*
- *RCA_4bit.v*
    - *4 bit ripple carry adder (1 gate delays)*
- *FullAdder.v*
    - *Full adder (1 gate delays)*
- *CSA_64bit_tb.v*
    - *Testbench for the carry select adder (1 gate delays)*
- *RCA_64bit_tb.v*
    - *Testbench for the ripple carry adder (1 gate delays)*

For the equivalent .v files with 0 gate delays, they each have an identical name to their 1 gate delay version, with a "_0" at the end.