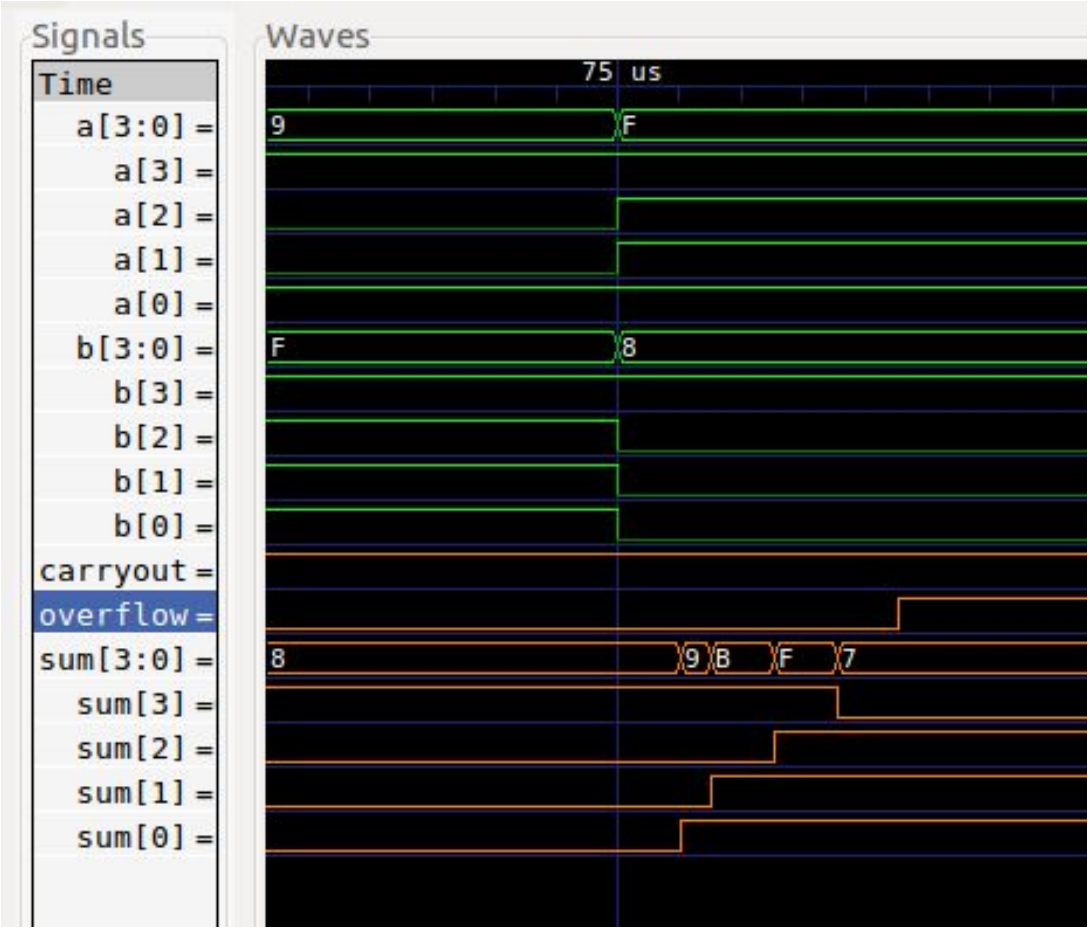
Coleman, Nick, & Yichen
09/26/2018

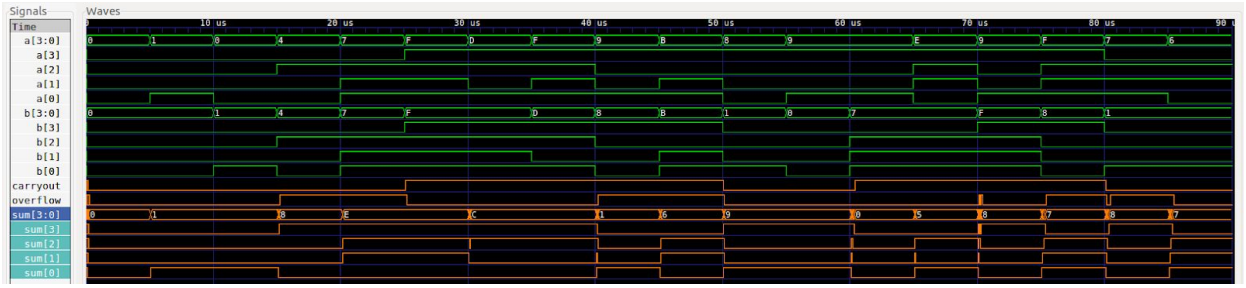
Lab 0

Simulation of Waveforms and Worst Case:

After simulating the full adder with the attached code, we simulated what the waveforms would look like. One of the tests we did is below.



This screenshot from GTKWave shows the worst case scenario for our outputs stabilizing after out inputs change, which ends up being 450 “time units”, as we set a delay of 50 “time units” for each gate. This was the worst timing we experienced, over any of our test cases, including several where the change propagated through all of the adders, then changed the overflow bit, which results in the longest time delay between the inputs changing and the outputs settling. Here’s the trace of our entire testbench simulation:



Initially, we were only checking for overflow by XORing sum[3] and the final carryout, which successfully finds an overflow if you're adding two positive or two negative numbers, but will also output a 1 if you add a negative to a positive number. So we added an additional NXOR gate from a[3] and b[3], then ANDed its output with the output of that XOR from sum[3] and carryout. This successfully displayed overflow when adding two positive or negative numbers together, and ensured that it doesn't trigger if you add a negative and a positive number together. *Note: We could also have XORed the carryin to the last adder and the final carryout to check for overflow, as stated in the Number Systems slides. We realized this after finishing the rest of the report.*

Truth Table Test Cases

For the truth table we designed, we created the following truth table with 18 cases in it.

```
comparch@comparchVM:~/Documents/Lab0$ ./adder
VCD info: dumpfile adder.vcd opened for output.
Four Bit Full Adder positive + positive Tests
 A   |  B   | Sum  | Carryout | Overflow | Sum Exp | Carryout Exp | Overflow Exp
0000 | 0000 | 0000 |    0     |    0     |  0000   | 0            | 0
0001 | 0000 | 0001 |    0     |    0     |  0001   | 0            | 0
0000 | 0001 | 0001 |    0     |    0     |  0001   | 0            | 0
0100 | 0100 | 1000 |    0     |    1     |  1000   | 0            | 1
0111 | 0111 | 1110 |    0     |    1     |  1110   | 0            | 1


Four Bit Full Adder negative + negative Tests
 A   |  B   | Sum  | Carryout | Overflow | Sum Exp | Carryout Exp | Overflow Exp
1111 | 1111 | 1110 |    1     |    0     |  1110   | 1            | 0
1101 | 1111 | 1100 |    1     |    0     |  1100   | 1            | 0
1111 | 1101 | 1100 |    1     |    0     |  1100   | 1            | 0
1001 | 1000 | 0001 |    1     |    1     |  0001   | 1            | 1
1011 | 1011 | 0110 |    1     |    1     |  0110   | 1            | 1


Four Bit Full Adder negative + positive Tests
 A   |  B   | Sum  | Carryout | Overflow | Sum Exp | Carryout Exp | Overflow Exp
1000 | 0001 | 1001 |    0     |    0     |  1001   | 0            | 0
1001 | 0000 | 1001 |    0     |    0     |  1001   | 0            | 0
1001 | 0111 | 0000 |    1     |    0     |  0000   | 1            | 0
1110 | 0111 | 0101 |    1     |    0     |  0101   | 1            | 0


Four Bit Full Adder Overflow/Carryout Tests
 A   |  B   | Sum  | Carryout | Overflow | Sum Exp | Carryout Exp | Overflow Exp
1001 | 1111 | 1000 |    1     |    0     |  1000   | 1            | 0
1111 | 1000 | 0111 |    1     |    1     |  0111   | 1            | 1
0111 | 0001 | 1000 |    0     |    1     |  1000   | 0            | 1
0110 | 0001 | 0111 |    0     |    0     |  0111   | 0            | 0
```

Explanation of test cases

The addition of two positive numbers test cases cover basic addition, including proving that the same sum is attained when the order of the numbers being added are changed and demonstrating what happens for an overflow error (we get a negative number).

The addition of two negative number test cases prove that we can add negative numbers relevant of order and they demonstrate what happens when an overflow occurs (we get a positive number).

The addition of a negative and a positive number shows what happen when there is no carryout and when there is a carryout; overflow is not possible in this case because a positive plus a negative cannot be higher than the highest positive or lower than the highest negative. The carryout is just due to the result being positive despite a negative number being added.

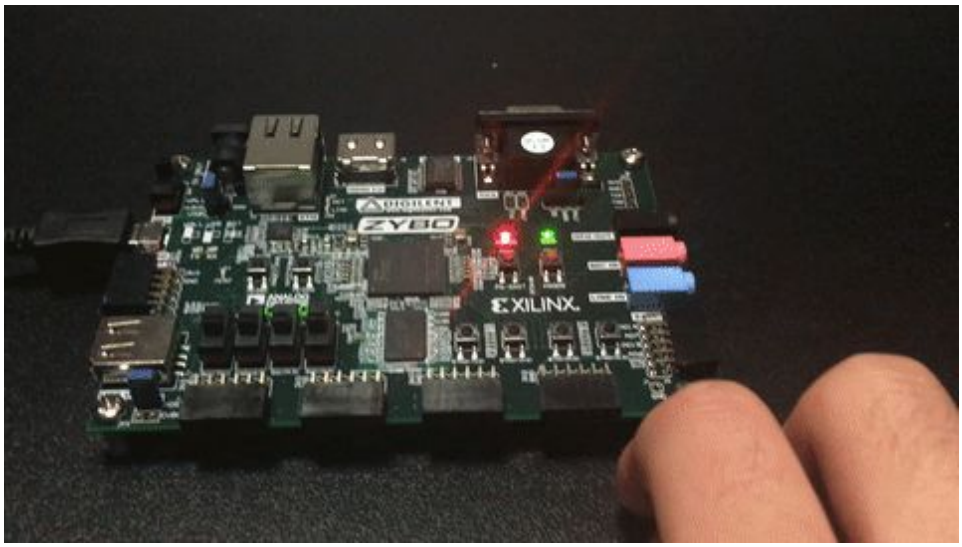
The final tests are related to overflow and carryout. There are four different cases related to both: one of the two is active, both are active, or neither are active. The four test cases picked each demonstrate one of the four different states to prove that they work.

Test Case Failures

We had one major failure in our test cases initially, which was due to overflow inaccurately recording due to a logic flaw. In order to remedy this, we changed the logic from an XOR taking

in the overflow and final bits to extend to checking to make sure that the bits in the most significant slot of the input test cases were the same before comparing overflow and carryout.

FPGA Test Summary



We tested our FPGA with 16 of our cases that we ran in the test bench. The reasoning behind choosing these test cases has been outlined above. All of the results came out as expected. The test above shows our test of A = 1111 and B = 1000. You can confirm with our above test bench that the FPGA showed the correct lights lit when we polled it for output.
(If you cannot see the GIF correctly, the link to our Google doc is https://docs.google.com/document/d/1Rc-CPYwRzufTj_RKRosjMLjHJeY0_yc-SnIiJCHRPzs/edit?usp=sharing)

Summary Statistics

DRC Violations

Summary: 1 warning
Implemented DRC Report

Timing — **Setup** | Hold | Pulse Width

Worst Negative Slack (WNS):	NA
Total Negative Slack (TNS):	NA
Number of Failing Endpoints:	NA
Total Number of Endpoints:	NA

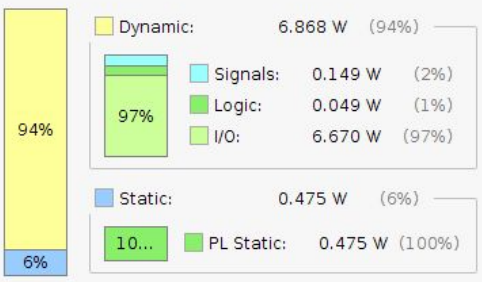
Implemented Timing Report

Utilization — Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	7	17600	0.04
FF	9	35200	0.03
IO	13	100	13.00
BUFG	1	32	3.13

Power — Summary | **On-Chip**



This is the summary statistic from our synthesized design. From the tables, we observe that our full adder design is relatively heavy on I/O (97% of power draw is due to I/O!).