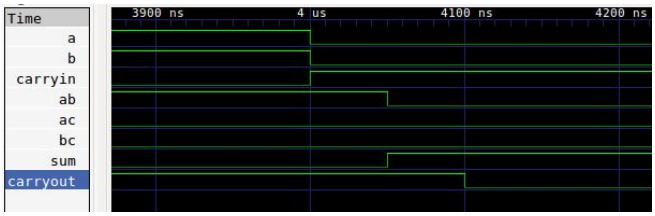
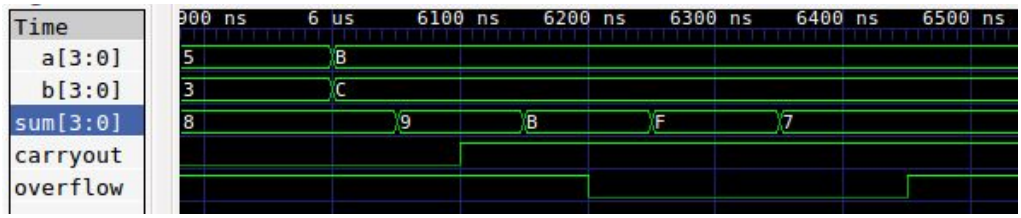
Delay and Waveform:

1. Single bit adder:
    a. Worst case delay:
        i. 2 gates before the carryout is calculated (only one xor gate for the sum, but everything waits on the carryout).
    b. Waveform:



2. Full Adder
    a. Worst Case delay:
        i. #100 * 4 1-bit-adders + #50 * 2 overflow logic gates = # 500 in total



The above image shows a near-worst case addition (1011 + 1100 = 0111 with overflow and carryout), where all but the final carryover calculation sum took two timesteps, and the overflow calculation took an additional two timesteps.

Test cases:

The first 9 test cases are chosen and tested in both verilog test bench (truth table) and on FPGA:

a. 0000 + 0000 = 0000
    ii. Make sure that 0 + 0 = 0, if this fails, something in the basic logic is wrong
b. 0010 + 0001 = 0011
    i. Again, basic logic, but with ones (no carryover yet)
c. 0011 + 0011 = 0110
    i. First test of carrying
d. 1111 + 1111 = 1110
    i. Testing negative numbers, but no overflow
e. 1110 + 1011 = 1001
    i. Testing different negative numbers with no overflow
f. 0101 + 0011 = 1000
    i. Testing positive numbers with overflow
g. 1011 + 1100 = 0111 (-5 + -4 = -9)

    i. Testing negative numbers with overflow

h. 1011 + 0100 = 1111 (-5 + 4 = -1)
    i. Positive plus negative = negative with no overflow

i. 1101 + 0101 = 0010 (-3 + 5 = 2)
    i. Positive plus negative = positive with no overflow

We think the above cases fully test the logic, so cases below are randomly selected to double test some of the above on the FPGA

j. 0011 + 0111 = 1010 (3 + 7 = 10)
    i. Positive plus positive with overflow and no carryout

k. 1101 + 1001 = 0110 (-3 + -7 = -10)
    i. Negative plus negative with overflow and carryout

l. 0011 + 0010 = 0101 (3 + 2 = 5)
    i. Positive plus positive, no overflow, no carryout

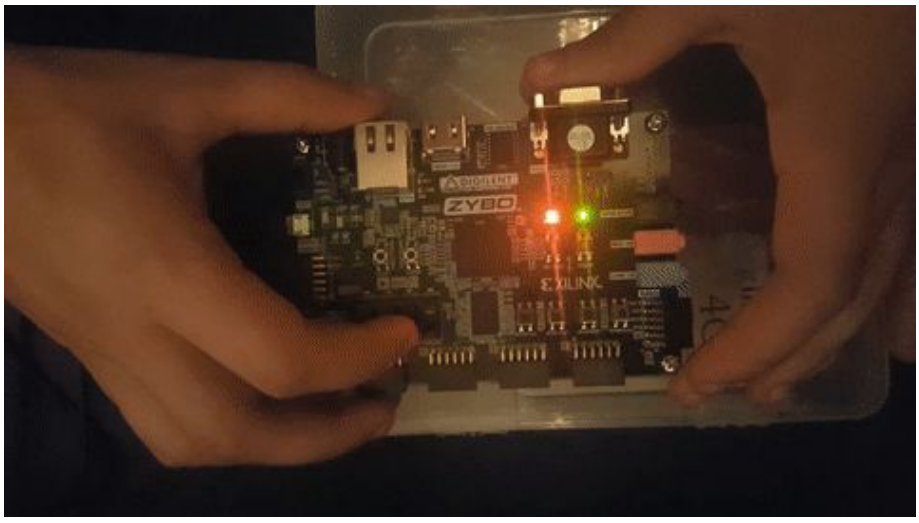
m. 1010 + 0010 = 1100 (-6 + 2 = -4)
    i. - + - without anything

n. 1001 + 1001 = 0010 (-7 + -7 = -14)
    i. Negative plus negative with overflow and carryout

o. 0011 + 1110 = 0000 (2 + -2 = 0)
    i. Negative plus positive, carryout but no overflow

p. 1001 + 0111 = 0000 (-7 + 7 = 0)
    i. Negative plus positive, carryout but no overflow

FPGA test



This shows our design adding 3 (0011) and 7 (0111), which adds to 1010, which is 10, read as a positive number, but -6 in the two's complement representation we are using, therefore the sum overflows. In the gif, 3 is set first, 7 second, at first the carryout/overflow setting is set, showing an overflow and no carryout. Then the mode is switched to sum, and the lights show 1010, then the mode is switched back to carryout/overflow.

We further tried all 16 cases listed in the previous section, and all of them displayed the correct result.

Errors and fixes:

2. Wire carry;
   We tried to use the same carry for all bits, which gave us xx (undefined outputs) in all but the first bit of the sum. We changed the program to use separate wires for carry1, carry2, carry3, and carryout, and the problem was solved.

3. 1101 +0101 (negative and positive), should not overflow, but failed our overflow test.
   Our original overflow logic was:
   overflow = (carryout xor sum[3])
   This evaluated to True because carryout was 0 and sum[3] was 1.

   Our original logic works for positive plus positive, or negative plus negative, but always fails for positive plus negative.
   To fix this, we changed the logic to:
   Overflow = (carryout xor sum[3]) and (a[3] xnor b[3]);
   We anded our original logic with (a[3] xnor b[3]), which is only true if the sign bits of the two inputs (a and b) are the same. This logic ensures that it will overflow only if the two numbers have the same sign. It will never overflow if the two numbers added have opposite signs.

Summary statistics analysis:

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

LUT is a lookup table, which we used 7, I think these allow us to perform our logic of addition. FF are flip flops, which we used 9, I think these allow us to push button and switch our output on LEDs. We used a lot of IO (input and output) because we had 4 buttons, 4 LEDs and 4 switches, as well as a clock. Lastly we used one BUFG, which is a global buffer. From a search online, “Global buffers are most commonly used for clock nets to provide the least amount of

skew possible between registers that are physically located large distances apart. You can also use them to provide quick access to control signals in high speed applications.” I don’t think our program requires one, but it might be the default one that’s always used for any program.

The timing here are NAs because we don’t have a clock.
Input and output took up the most power, followed by signals and then logic, both contributing relatively little power consumption.