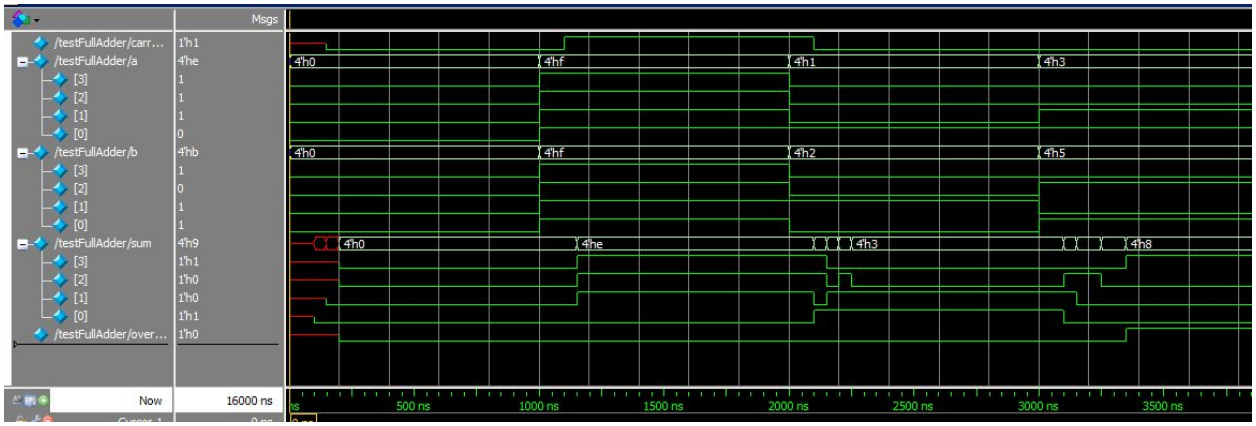
# Lab 0: Final Report

Jessica Diller, Marie-Caroline Finke, and Griffin Tschurwald

## Waveforms




Since we assumed a delay of 50 units of time per gate the worst case delay would be 500 units of delay. This is calculated as the worst case delay for each adder module is 3 gates for the first, 2 for the second three and 1 for the XOR gate calculating overflow.

## Test Case Strategy: Failures and Design Changes

We began our tests in modelsim as we were creating our four-bit adder. We thought to begin with a simpler test and then build up in complexity for further tests. We also paid attention to the changes we made in our code from the previous one-bit adders and made sure to test those areas specifically.

1) 0000 + 0000 = 0000: This tested the initial functioning of the system and whether or not it would break with an input. It worked.
2) 0101 + 0011 = 1000: This tested the ability of our wires between adders to carry bits from one to another. It worked.
3) 1001 + 1110 = 1001 (1): This tested the ability for our system to handle a carryout and overflow. This one actually gave an output of 0111 (1). We looked at the circuit we created and realized that implementing our carryout using a not gate and an and gate was incorrect. We needed an XOR gate instead, as our

previous solution only worked properly if the final carryin was false and the final carryout was true, but would fail if the final carryin was true and the final carryout was false. We need it to return overflow as true in both of those cases.

4) 1110 + 1100 = 1010 (1): This tested the ability for our system to handle just carryout. It worked.
5) 1001 + 1110 = 1001 (1) We got 0111 (1) with an overflow of one, which isn't correct, because we hadn't fully fixed the error from test case #3. Upon mathematical analysis of this error, we thought it would have to do with the first adder carrying out a one instead of a zero bit. The code had been changed at this adder more than the others; we took all mention of a "carryin" out of the code. To test this change, we inserted a statement "assign carryin0 = 0;" so that it would always be 0. After changing this, all of our test cases worked and we were good to go.

For the sixteen test cases for our FPGA, we decided to use the ones above, but add some more. Our initial thoughts were to test three different adding scenarios: positive + positive, positive + negative, and negative + negative number. Another category of importance was checking carryouts and overflow. We thought we'd try each one of the adding scenarios with three other categories: no carryout and no overflow, carryout and no overflow, and carryout and overflow. Those nine situations combined with the zero cases of 0000 +0000 and 1111 + 1111 would have given us a 11 decided test cases. We thought then to randomized the rest of them would give us a picture of anything else we hadn't thought about.

Upon further thought, we discovered that not all the situations we planned could ever exist. Positive + positive doesn't allow for carry out, negative and positive + negative doesn't allow for overflow, and negative + negative needs to have both carry out and overflow (or neither). That brought our total down to eight test cases. We chose to randomize the rest of them (eight). See the truth table from ModelSim that is listed below.

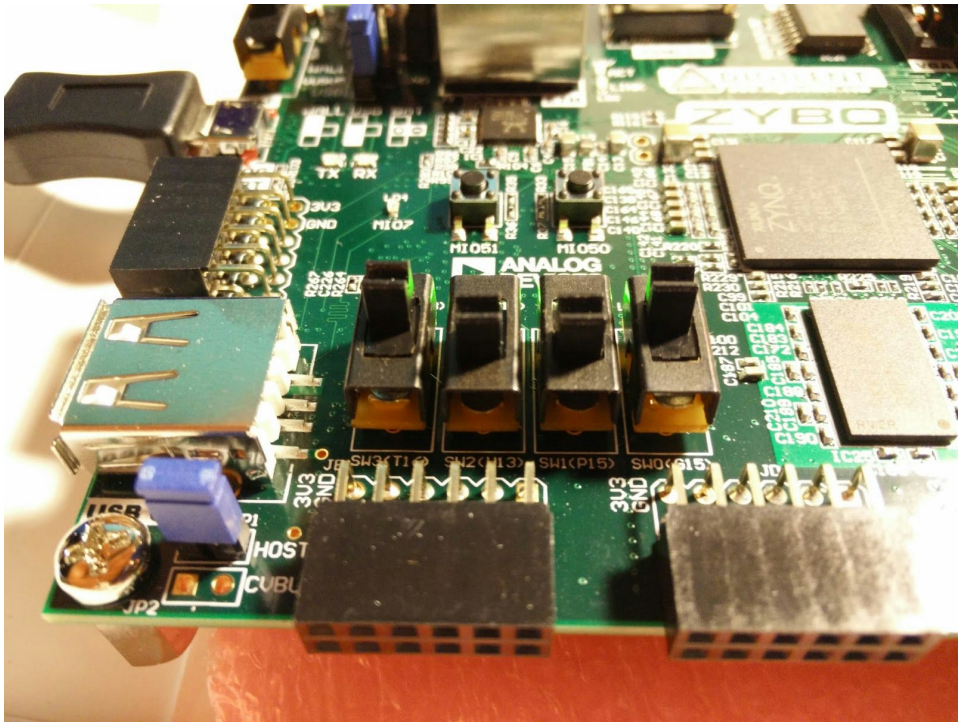
```
#    A     B   |  Sum  Cout| Overflow | Expected Output (carryout) | Expected overflow
#  0000  0000 |  0000  0  |    0     | 0000 (0) | 0
#  1111  1111 |  1110  1  |    0     | 1111 (1) | 0
#  0001  0010 |  0011  0  |    0     | 0011 (0) | 0
#  0011  0101 |  1000  0  |    1     | 1000 (0) | 1
#  1111  1110 |  1101  1  |    0     | 1101 (1) | 0
#  1011  1100 |  0111  1  |    1     | 0111 (1) | 1
#  1011  0011 |  1110  0  |    0     | 1110 (0) | 0
#  1101  0110 |  0011  1  |    0     | 0011 (1) | 0
#  1110  0110 |  0100  1  |    0     | 0100 (1) | 0
#  0100  1110 |  0010  1  |    0     | 0010 (1) | 0
#  1100  0011 |  1111  0  |    0     | 1111 (0) | 0
#  1000  0101 |  1101  0  |    0     | 1101 (0) | 0
#  0011  0100 |  0111  0  |    0     | 0111 (0) | 0
#  0101  0111 |  1100  0  |    1     | 1100 (0) | 1
#  1000  1111 |  0111  1  |    1     | 0111 (1) | 1
#  1110  1011 |  1001  1  |    0     | 1001 (1) | 0
```

In addition to the implementation required of us, we added buses to our verilog code. This gave us multiple errors. We had to research the correct way of bus implementation instead of having four different input variable names in both the .v file and the .do file. Online, we found how to write the .v file code, but we had to find a NINJA to learn that, in the do file, we just had to define the letter name of the bus, not the inner bracket indexes.

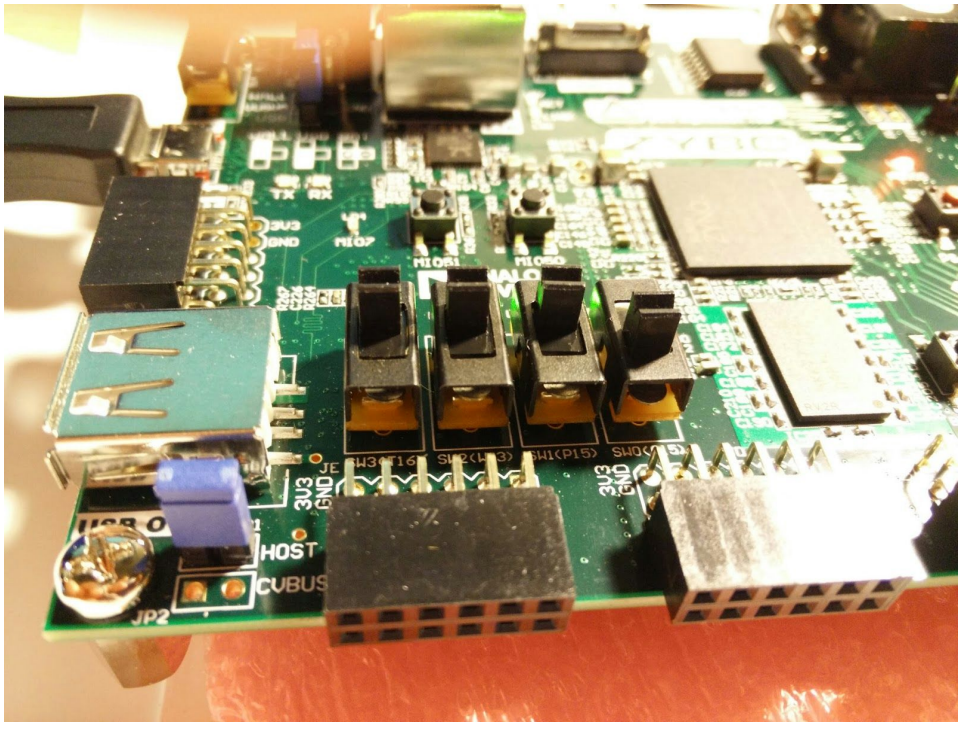
# FPGA results

In order to test our FPGA board, we had to properly set up our .xdc file to allow use of the 4 leds, 4 buttons, 4 switches, and the clock. Then we just had to change the syntax in our FullAdder4bit() call to match the syntax in the lab0_wrapper.v file. We wound up using the default lab0_wrapper.v file instead of making our own. Below are pictures of us testing the addition of b1001 and b1110, which should result in a sum of 0111 with a carryout of 1 and an overflow of 1. In addition, we tested our other 15 test cases successfully on our fpga board.

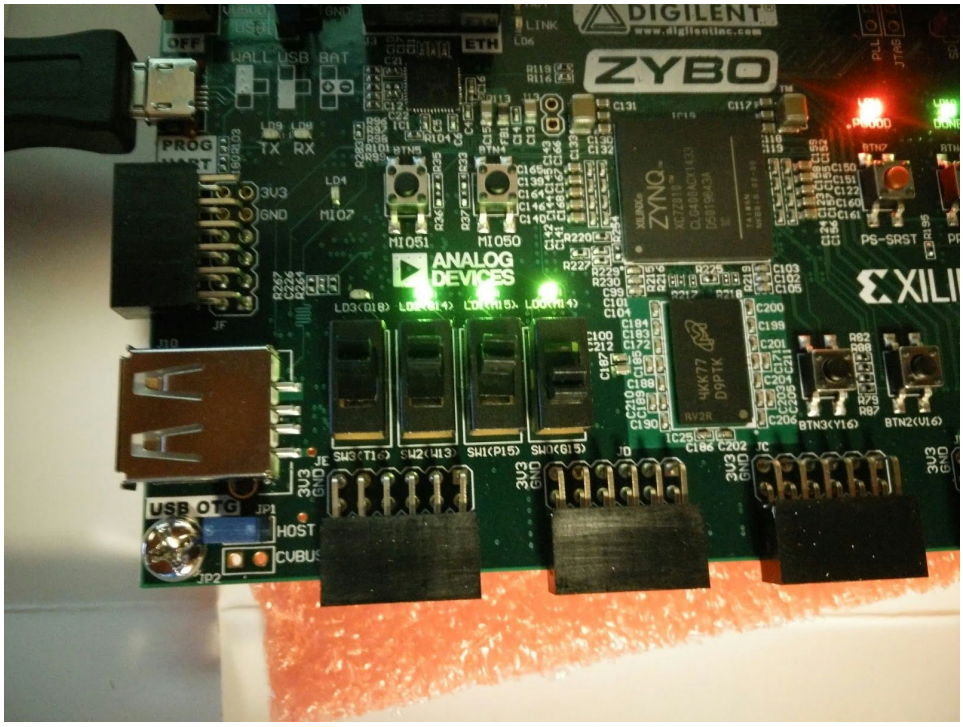
Setting input to: 1001



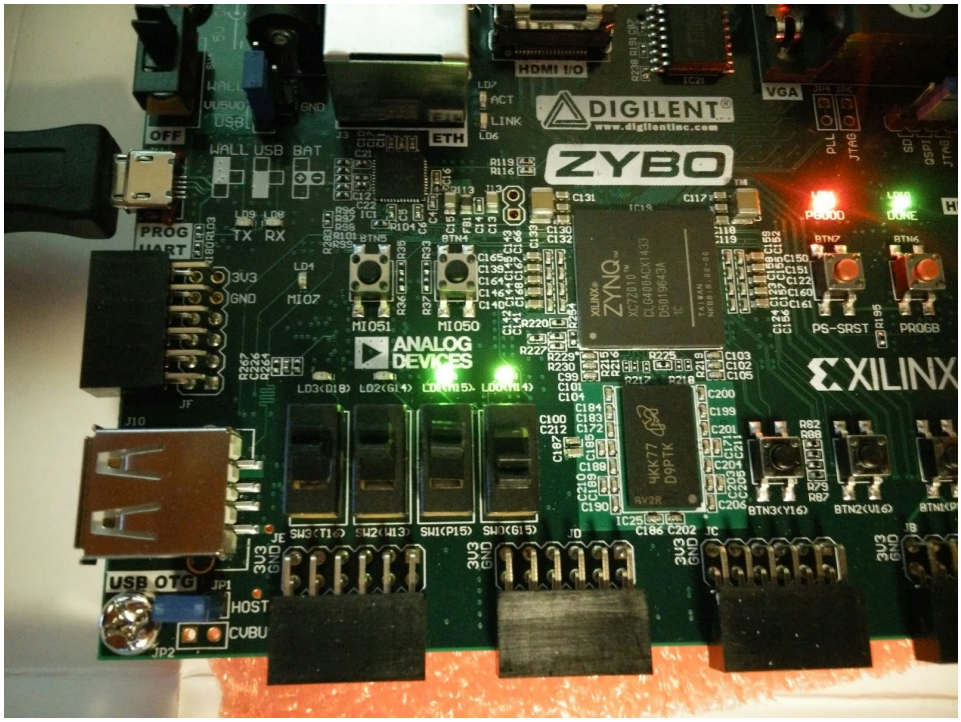

Setting input b to: 1110



Using button 3 to read the sum from the four LEDS of 0111:




Using button four to show the carryout on LED0 and the overflow on LED1:



# Design Statistics

After implementing and running our verilog files on our Zybo board, we looked at our summary statistics for our design. Below are the numbers captured from our run. We weren't exactly sure what some of these statistics were referring to, so we made our best approximate guess of what they are.

*Total on-chip power:* 0.113W. This is the power used for the implementation of our design.

*FF:* We used 9 of 35200 for a utilization percentage of .03%. This appears to be the number of flip-flops used in the design.

*LUT:* We used 9 of 17600 for a utilization percentage of .05%. This appears to be the number of lookup tables we used while implementing our design.

*I/O:* We used 13 of 100 for a utilization percentage of 13%. This appears to be the number of input and output devices we used. We used 4 leds, 4 switches, 4 buttons, and the usb input port for a total of 13 I/O.

*BUFG:* We used 1 of 32 for a utilization percentage of 3.12%. This appears to be a clock buffer that we only used one of.