Computer Architecture Fall 2016

Lab 0 Report: Full Adder on FPGA

Ian Paul, Jee Kim, Shruti Iyer

Introduction

For this lab, we implemented 4 bit full adder on FPGA. We built on the structural full adder module we have previously constructed to create a 4 bit full adder. The first part of the lab consisted of verifying FPGA tool chain and understanding the lab0_wrapper.v code. Then, we coded 4 bit full adder and testbench generator. Lastly, we implemented the code on FPGA and checked the test cases to see if it was functioning as we expected it to.

Code

We created structural 4bit full adder with delay of 50 units (50ns) for each gate (adder.v) and code for generating all test bench cases (fulladder.t.v).

For generating all test cases, we used a for loop to loop through a and b from b00000 to b01111. The extra 4th bit was necessary to terminate the for loop. In each loop, we added a delay of 1000 units (1us) when it run the innermost for loop so that the system could stabilize once A and B changed.

Waveforms

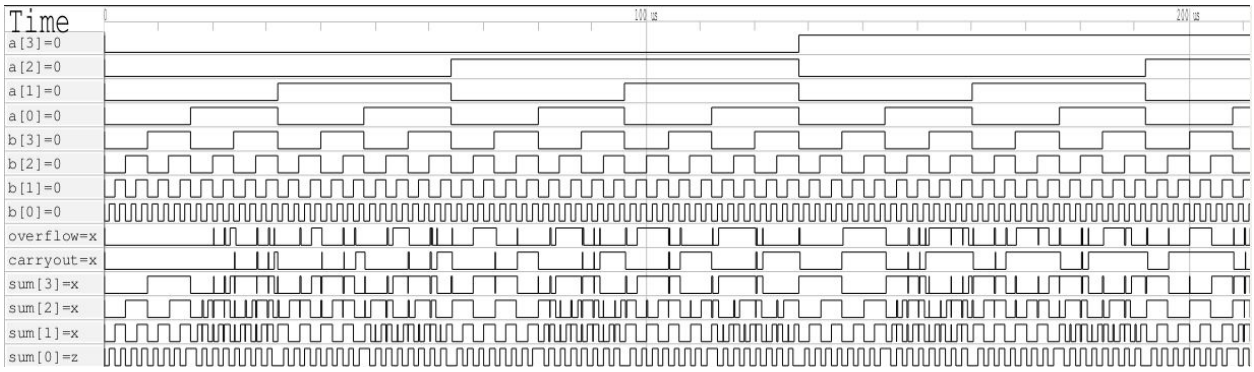


Figure 1. Full waveform of all 256 cases

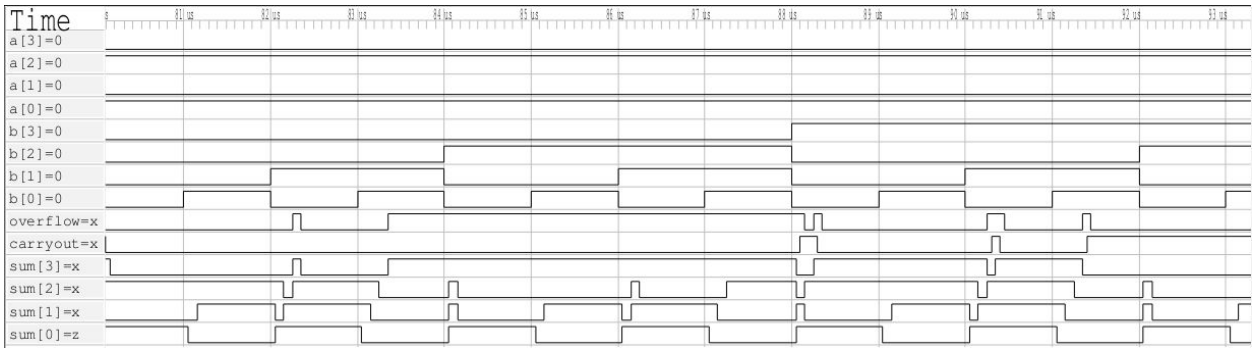


Figure 2. Zoomed in version of waveforms from 80us to 96us

Figure 1 shows the full waveform of all 256 cases we've tested. Figure 2 shows a zoomed in version of the waveform from 80us to 96us. From Figure 2, we can see that there are glitches in waveform which it the artefact of delays from the gates.

Worst Case Delay

The worst case delays happen when everything digit in the sum changes from 1 to 0 or 0 to 1. From the test bench, we read that the worst case delay is 450 nanoseconds. One of the worst case delays from the test bench is shown in Figure 3.

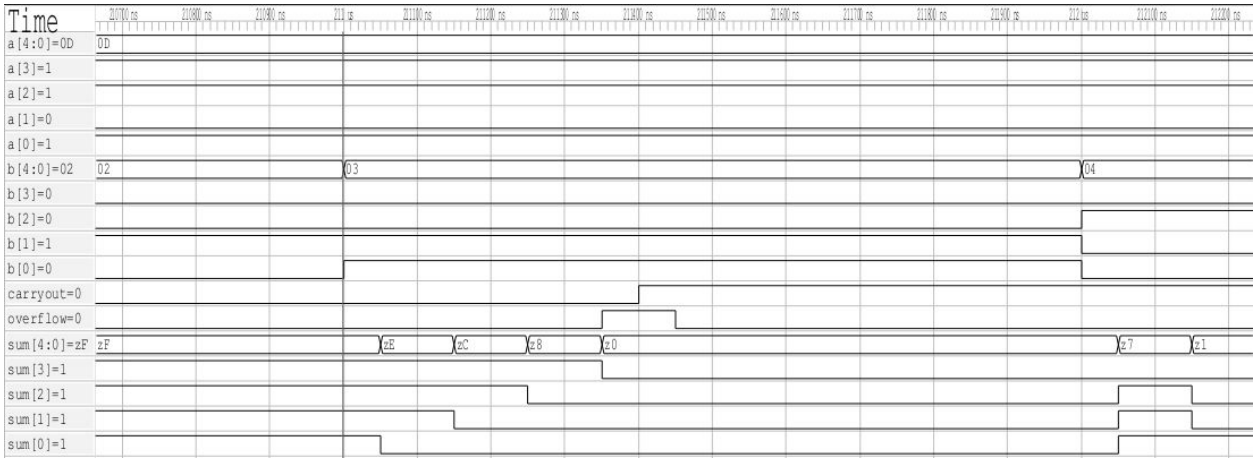


Figure 3. Showing one of the worst case delays that happened at 211us. The carry out took 400ns to switch and there is a glitch at overflow happened from 350ns to 450ns since the b changed at 211ns.

The longest delay path for each adder is 2 gates (one AND gate and one OR gate for computing the carryout). This takes 100ns since each gate was set to have delay of 50ns. The 4 bit full adder has to go through 4 adders which gives delay of 400ns. We can see that the carryout has stabilized after 400ns. However, overflow is determined afterwards from XOR of carry in and carry out of the last sum digit. The XOR gate has 50ns delay and the worst delay takes 450ns to settle.

Implementing on FPGA

After checking that the code was functional, we went on to implement our code on FPGA. We could not test all 256 cases from the switches and buttons on FPGA so we chose 16 test cases we wanted to test on FPGA.

Test Case Strategy

We wanted our test cases to capture both the areas we expect possible failure and success. We looked for cases where we add two positives (with and without overflow), two negatives (with and without overflow), a positive and a negative, and zero to values. We also checked to make sure the carry out was behaving as expected.

List of Test Cases

a	b	sum	cout	overflow
0000	0000	0000	0	0
0111	0111	1110	0	1
1001	1001	0100	1	1
0111	1001	0001	0	0
0011	0011	0110	0	0
0110	0001	0111	0	0
1111	0001	0000	1	0
1101	1101	1010	1	0

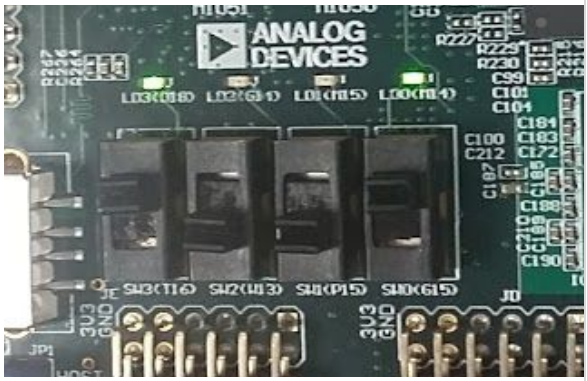
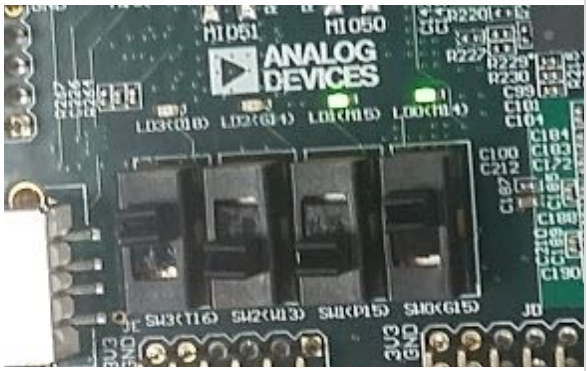
a	b	sum	cout	overflow
1011	1101	1000	1	0
0001	0001	0010	0	0
0101	1010	1011	0	0
0110	0011	1001	0	1
1111	1111	1110	1	0
1110	1101	1011	1	0
1100	1100	1000	1	0
1000	1111	0100	1	1

Test Bench Failures

Initially, we detected overflow by XORing the most significant sum (the last sum computed) with the last carryout. However, when we added -1_{10}(b1111) with 2_{10}(b0011), we got 1_{10}(b0001) with carryout 1. XOR of the most significant sum, b0, and the carryout, b1, gave b1 although there was no overflow. From this case failure, we learned that we should calculate overflow by XORing the last carryout with the last carryin. We corrected the 4 bit full adder code accordingly and then tested our list of test cases and generated our waveforms.

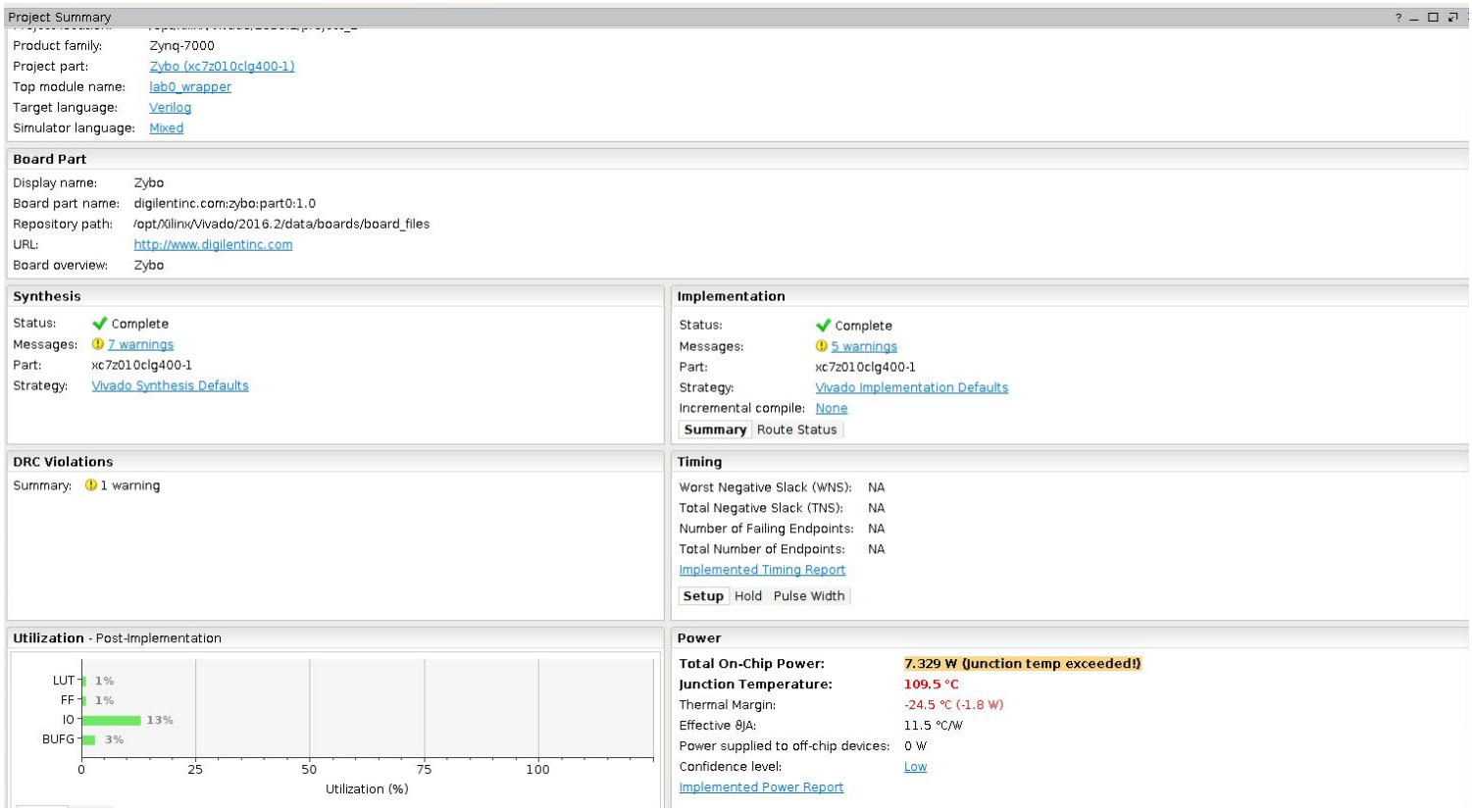
FPGA Testing

We tested all of the test cases in the table above on the FPGA. All of our test cases gave the result we expected and we are confident that it will work for all 256 cases. Below are pictures of one of the test cases.

	
1. We started out with a, b, and the sum all at 0	2. First we set a to 1001
	
3. We set b to 1001, and the sum showed as 0010	4. Both the overflow and carry out showed as 1

Summary statistics

Test Cases: 16
Tests passed: 16
Percent of tests passes: 100%



Because the circuit is rather basic we can't analyze the summary statistics too much. Performance, energy, and area are not being reported in a useful fashion in this summary. What is interesting is the utilization section, showing that most of what we use the IO on the chip. We also use lookup tables, flip flops, and global buffers some, but much less than the IO.

Resources

[1] https://sites.google.com/site/ca16fall/resources/fpga
[2] https://github.com/CompArchFA16/Lab0
[3] http://sandbox.mc.edu/~bennet/cs110/tc/orules.html
[4] http://teaching.idallen.com/cst8214/08w/notes/overflow.txt