Comp Arch: Lab 0
9/23/2018
Vicky McDermott
Minju Kang
Allison Basore

Comp Arch: Lab 0

For this lab, we made a four bit full adder simulation and on our FPGA. The purpose of this lab is to create and test a four bit full adder using four of our previously made full adders, create test cases for various combinations, learn how to use an FPGA, and test our four-bit adder with the FPGA.

Simulation

Previously, we had each built an adder module in structural Verilog and tested it with the *iverilog* simulator. For this lab, we used four of those adders combined to make a four bit adder. Each adder is for calculating the addition of a digit place of a four digit binary number. In order to calculate overflow, we also had an XOR gate from the most signficant and second to most significant digits. This overflow variable is for identifying if the sum is an overflow of the four bits we have to work with for our sum into the signed bit. If overflow is true, we cannot count on the signed sum result being correct. In unsigned arithmetic on the other hand, the carryout variable identifies if there is a carryout when adding the two most significant bits. If carryout is true, we cannot count on the unsigned sum result being correct. Figure 1 and 2 shows our circuit diagram:

Figure 1: Full Adder Circuit Diagram

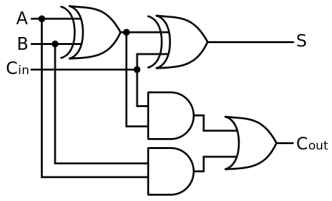
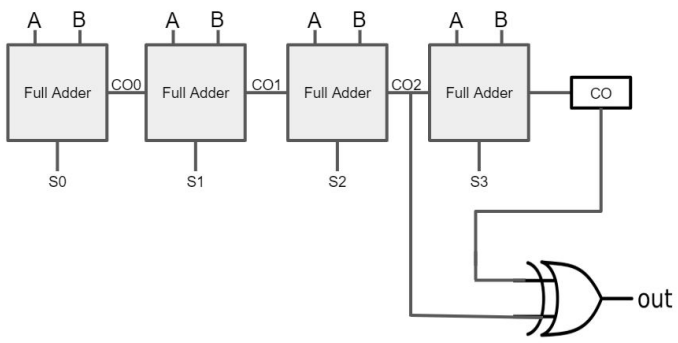


Figure 2: 4 Bit Adder Circuit Diagram

Comp Arch: Lab 0
9/23/2018
Vicky McDermott
Minju Kang
Allison Basore



*There is a 50 μs delay between each gate.

Since there are a total of 256 possible test cases, we chose to pick 16 that would test different capabilities of our adder including options where there is or is not carry out and where there is or is not overflow. The table in Figure 2 shows our test expected outputs and actual outputs on our test bench.

A3	A2	A1	A0	B3	B2	B1	B0	Overflow	Expected Overflow	Sum3	Sum2	Sum1	Sum0	Sum Should Be	Carry Out	Expected Carry Out
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000	0	0
1	1	1	1	1	1	1	1	0	0	1	1	1	0	1110	1	1
0	0	1	1	0	0	1	1	0	0	0	1	1	0	0110	0	0
1	1	0	0	1	1	0	0	0	0	1	0	0	0	1000	1	1
0	0	0	1	0	1	0	0	0	0	0	1	0	1	0101	0	0
0	1	0	1	1	0	1	0	0	0	1	1	1	1	1111	0	0
0	1	0	1	0	1	0	1	1	1	1	0	1	0	1010	0	0
1	1	0	1	0	0	1	1	0	0	0	0	0	0	0000	1	1
1	0	1	0	1	1	0	1	1	1	0	1	1	1	0111	1	1
1	0	1	0	1	1	0	0	1	1	0	1	1	0	0110	1	1
0	1	0	0	1	1	0	1	0	0	0	0	0	1	0001	1	1
1	0	0	1	0	1	1	0	0	0	1	1	1	1	1111	0	0
1	1	0	1	0	1	1	1	0	0	0	1	0	0	0100	1	1
0	1	1	1	1	0	0	1	0	0	0	0	0	0	0000	1	1
0	0	0	1	1	1	1	1	0	0	0	0	0	0	0000	1	1
1	0	1	1	1	1	0	1	0	0	1	0	0	0	1000	1	1

Figure 3: 16 Test cases with our four bit adder

Several of these test cases worked on our first attempt. To cover our combinations, we needed some that would result in overflow, some that would result in carry out past the most significant digit, and some with extreme values. An example of an extreme would be 0000 with 0000 or 1111 with 1111. Ten of our cases had a resulting carryout and 3 of our cases had overflow. We also covered the previously mentioned extremes. As shown in Figure 3, all of our test cases match up with the expected outputs.

Furthermore, in Figure 4, we show the waveforms of our 16 tests separated by a nanosecond delay. Later in the report, we discuss this timing in depth.

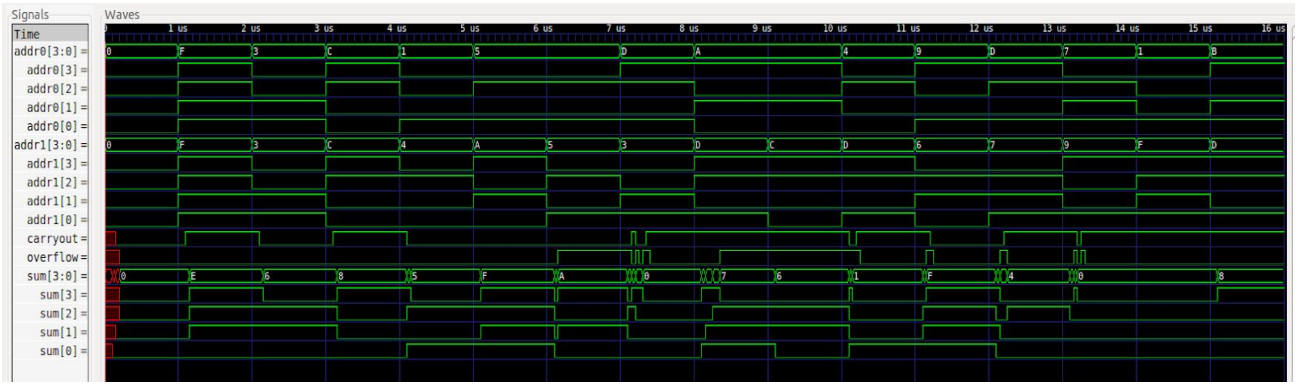


Figure 4: gtkwave Simulation of 4 bit adder

At the beginning we had several failures that we had to debug. Originally, we had ordered the bits of the input and output to be 0→3. However, we realized that the the most significant digit is actually 3, and therefore we should index from 3→0. Another problem we had was fully understanding how to calculate with two's complement. We sought outside resources and were able to gain ample practice by manually calculating our 16 test cases.

After making sure our FPGA board was functional by following the FPGA tutorial that was given, we were able to use the adder wrapper to test our four bit adder on the FPGA board. Below are the images taken at each point of setting up a test for 1100 added to 1100. The output for this should be low overflow and high carryout with a sum of 1000.

Comp Arch: Lab 0
9/23/2018
Vicky McDermott
Minju Kang
Allison Basore

Setting the first input to 1100

Setting the second input to 1100

Showing the output sum to be 1000

Showing carryout value to be high and the overflow to be low

Comp Arch: Lab 0
9/23/2018
Vicky McDermott
Minju Kang
Allison Basore

Propagation Delay for 1-Bit Full Adder (Unit delay for each gate is 50μs)

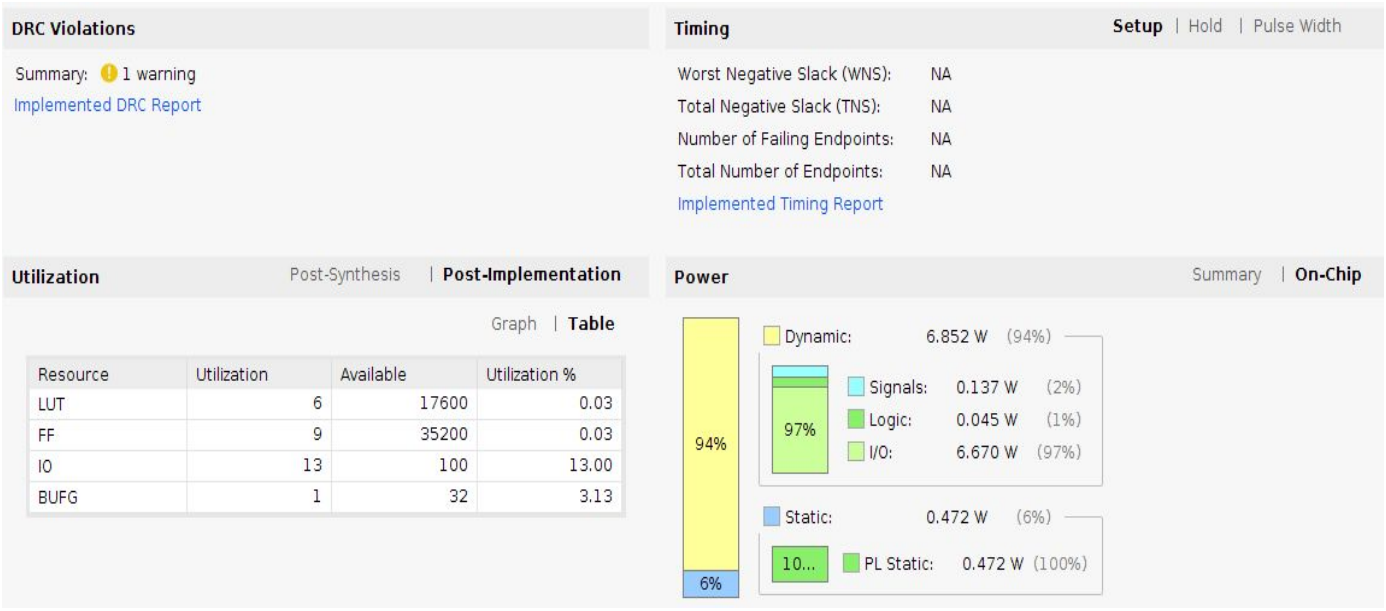
	A	B	Cin
Sum	2	2	1
Cout	3	3	2

Propagation delay is calculated by adding up the delays from each of the gates that you pass through to drive a result from an input. Since the delay considers for the worst case, you want to have the signal goes through the maximum number of gates. For example, from input A to Sum, the signal passes through 2 XOR gates. Since each XOR gate has unit delay of 50μs, for A to Sum, the total propagation delay will be 50μs * 2 = 100μs. Such delay can be eliminated by implementing a Look Up Table, which we will discuss in later labs.

Propagation Delay Chart for 4-Bit Full Adder (Unit delay for each gate is 50μs)

	A/B0	A/B1	A/B2	A/B3
S0	2	-	-	-
C0	3	-	-	-
S1	3 + 1 = 4	2	-	-
C1	3 + 2 = 5	3	-	-
S2	(3 + 2) + 1 = 6	3 + 1 = 4	2	-
C2	(3 + 2) + 2 = 7	3 + 2 = 5	3	-
S3	(3 + 2 + 2) + 1 = 8	(3 + 2) + 1 = 6	3 + 1 = 4	2
Carry Out	(3 + 2 + 2) + 2 = 9	(3 + 2) + 2 = 7	3 + 2 = 5	3

Figure 5: Summary of Vivado Statistics



The summary statistics shown above, shows that IO has the largest utilization because the bulk of this lab was sending inputs and receiving resulting outputs.