CompArch: Lab 0

Brenna Manning, William Saulnier, Ziyu (Selina) Wang

10-1-15

1 Waveforms and Worst Case Delay

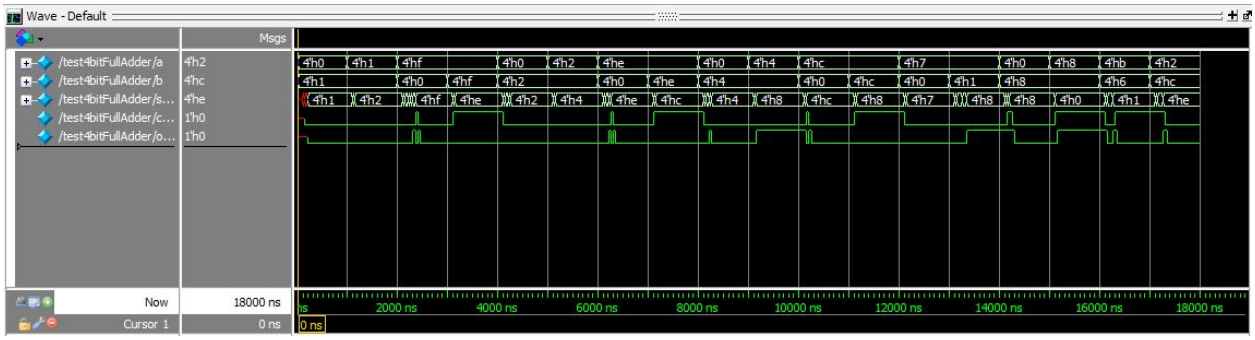

Figure 1: Waveforms showing the full adder stabilizing after changing inputs

The small glitches at the beginning of the sum, carry out, and overflow waveforms are caused by delays between the time the inputs are initialized and the time sum, carryout, and overflow are computed. The total time 18000 ns is because we are running 18 tests and each if them has a 1000 ns of delay. The small glitches in middle of the waveforms are caused by the propogational delays of the progression of inputs through various logic gates and then finally reaching the calculation of sum, carryout, and overflow.

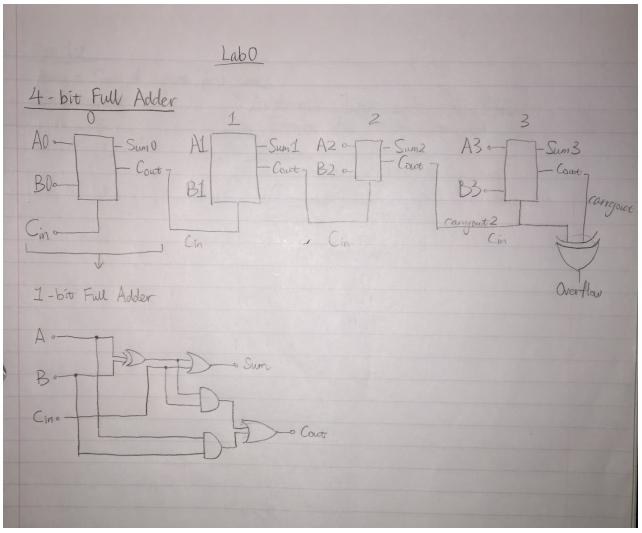

Figure 2: Schematic of the 4 bit adder with overflow and the 1 bit adder

The worst case delay would be 13 units of delay which is the same as 13 * 50 units of time = 650 units of time. Since the worst case delay for the 1 bit adder is 3 units of time (After an input is initialized, it has to pass through a xor gate, an and gate, and an or gate to reach carry out) and there are four 1 bit adders chained together, the worst case delay for the last carryout would be 12 units of time. And then it takes another xor gate to reach overflow which sums up to 13 units of time.

2 Test Bench Methodology

```
# Loading work.test4bitFullAdder
# Loading work.FullAdder4bit
# Loading work.structuralFullAdder
# Test Adder 0 Pos:
# a     b    | sum    carryout   overflow| Expected Output
# 0000  0001 | 0001   0          0       | 0001    0     0
# 0001  0001 | 0010   0          0       | 0010    0     0
# Test Adder 0 Neg:
# a     b    | sum    carryout   overflow| Expected Output
# 1111  0000 | 1111   0          0       | 1111    0     0
# 1111  1111 | 1110   1          0       | 1110    1     0
# Test Adder 1 Pos:
# a     b    | sum    carryout   overflow| Expected Output
# 0000  0010 | 0010   0          0       | 0010    0     0
# 0010  0010 | 0100   0          0       | 0100    0     0
# Test Adder 1 Neg:
# a     b    | sum    carryout   overflow| Expected Output
# 1110  0000 | 1110   0          0       | 1110    0     0
# 1110  1110 | 1100   1          0       | 1100    1     0
# Test Adder 2 Pos:
# a     b    | sum    carryout   overflow| Expected Output
# 0000  0100 | 0100   0          0       | 0100    0     0
# 0100  0100 | 1000   0          1       | 1000    0     1
# Test Adder 2 Neg:
# a     b    | sum    carryout   overflow| Expected Output
# 1100  0000 | 1100   0          0       | 1100    0     0
# 1100  1100 | 1000   1          0       | 1000    1     0
# Test Adder 3 Carryout Pos + Pos:
# a     b    | sum    carryout   overflow| Expected Output
# 0111  0000 | 0111   0          0       | 0111    0     0
# 0111  0001 | 1000   0          1       | 1000    0     1
# Test Adder 3 Carryout Neg + Neg:
# a     b    | sum    carryout   overflow| Expected Output
# 0000  1000 | 1000   0          0       | 1000    0     0
# 1000  1000 | 0000   1          1       | 0000    1     1
# Test Adder 3 Carryout Neg + Pos:
# a     b    | sum    carryout   overflow| Expected Output
# 1011  0110 | 0001   1          0       | 0001    1     0
# 0010  1100 | 1110   0          0       | 1110    0     0
```

For our test bench, we saught to select tests that would both comprehensively test the most common "trap" cases, as well as isolate causes of error inside of our circuit system. Therefore, even though it was not strictly necissary, we selected tests that would isolate each adder in turn, as well as verifying the proper overflow and carry behaviors. We therefore split our tests into five sections: Testing Adder 0, Adder 1, Adder 2, Adder 3 when adding two positives, Adder 3 when adding two negatives, and Adder 3 when adding a positive and a negative. By isolating the testing to each adder in turn, we were able to build on top of successful previous tests in order to test further complicated circuits. An error in the Adder 1 test, which builds off a successful Adder 0 test, would help to isolate our error case to that specific adder as opposed to having to plumb through the whole circuit.

3 Test Errors

We encounted no errors when performing our test cases. We had confidence in our individual adder implementation which was fully tested as part of HW1, so the careful chaining of them together for our Lab seemed to work effectively. The only point of challenge would have been ensuring that the overflow was calculated correctly. With some colaberative thinking, we were able to arrive at the correct solution on a first try, which resulted in no errors in our implementation, which is shown through our test bench.

4 A Summary of Testing Performed on the FPGA Board

We programmed our FPGA Board with the `adder.v` and the `lab0_wrapper.v` verilog files as well as the default constraint file, with the lines setting the components we would use on the board uncommented. The switches on the FPGA board were set to represent 4-bit binary numbers which were saved to represent "A" and "B" by pressing button 0 and button 1 respectively. Then, when button 2 was pressed, the LEDs on the board would display the value of the sum, and when button 3 was pressed they would display the carry-out and overflow of the addition.

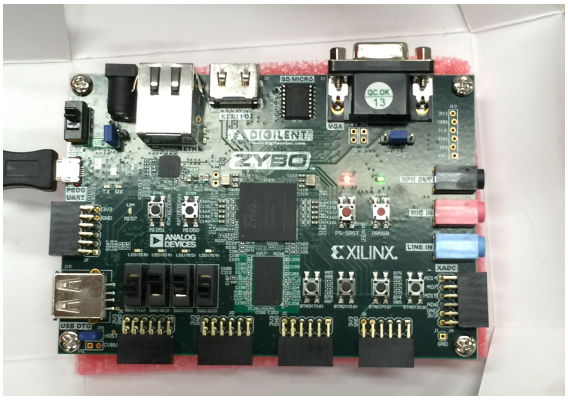

Figure 3: Setting A to be 0110

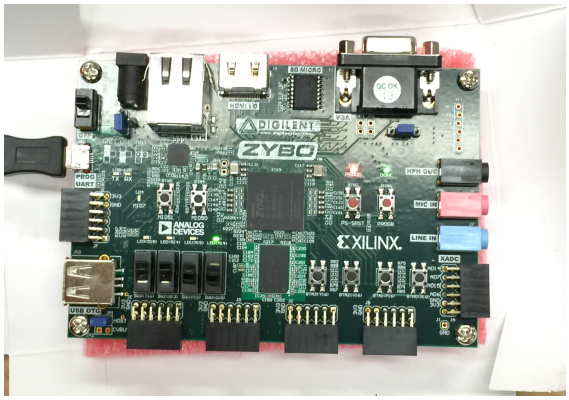

Figure 4: Setting B to be 1100

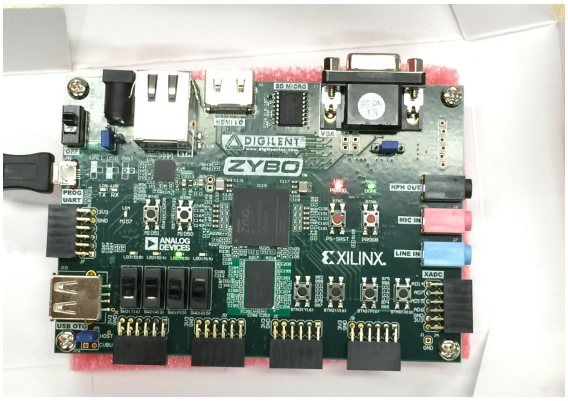

Figure 5: The sum is 0010

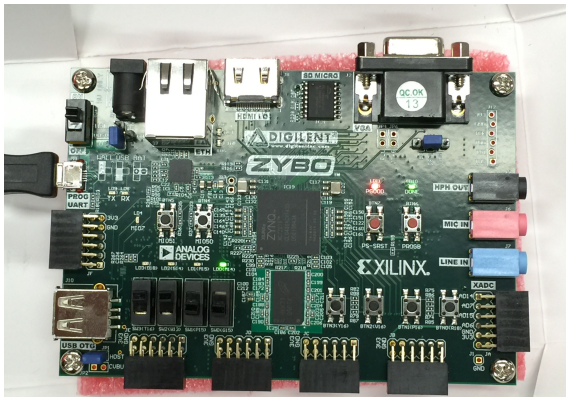

Figure 6: No carry out, overflow of 1

The chart below shows the 16 cases we tested on the FPGA board as well as the results it displayed.

Inputs		Outputs		
A	B	SUM	Carry-Out	Overflow
0001	0001	0010	0	0
0001	0000	0001	0	0
1111	0000	1111	0	0
1111	1111	1110	0	1
0010	0001	0011	0	0
0011	0010	0101	0	0
1110	1111	1101	0	1
1110	1110	1100	0	1
0110	0001	0111	0	0
0110	0010	1000	1	0
1011	1111	1010	0	1
1001	1110	0111	1	1
0110	1100	0010	0	1
0111	1000	1111	0	0
0101	0011	1000	1	0
1001	1110	0111	1	1

The results of each of these tests were exactly what we had expected. All of the sums are accurate, and all of the carry-out and overflow values are accurate. Similarly to our test benches, we chose test cases we knew would be comprehensieve enough to catch any likely errors. We chose to test some cases that we knew would result in no carry-out or overflow, some that would result in just carry-out, some just overflow, and some that would result in both. We tested cases that were interesting to us, such as those where the sign was flipped. We chose to test some that were the same as we used in our Verilog test bench, as well as some that were different. In every case, no matter what inputs we chose, the board behaved as expected, therefore our testing has concluded that oour FPGA Board works exactly as we would like it to.

5 Summary Statistics of Synthesized Design

From the Vivado project summary, we observe the following:

5.1 Timing

Overall, the clock period was 8.000 ns.

More timing information:

Setup	
Worst Negative Slack (WNS)	6.465 ns
Total Negative Slack (TNS)	0.000 ns
Hold	
Worst Hold Slack (WHS)	0.515ns
Total Hold Slack (THS)	0.000 ns
Pulse Width	
Worst Pulse Width Slack (WPWS)	3.500 ns
Total Pulse Width Negative Slack (TPWS)	0.000 ns

5.2 Utilization

The chart below shows the utilization of resources:

Resource	Utilization	Available	Utilization %
FF	9	35200	0.03%
LUT	9	17600	0.05%
I/O	13	100	13.00%
BUFG	1	32	3.12%