

CompArch Lab 0

Logan Sweet & Maggie Jakus

September 27, 2017

1 Introduction

An adder is a type of snake native to Europe. Its diet consists of small mammals, reptiles, and insects.

In this lab, we used the adder code developed in HW2 to create a four-bit full adder. We connected four one-bit full adders in series, and kept 50ns delays on all of the gates.

2 Waveforms

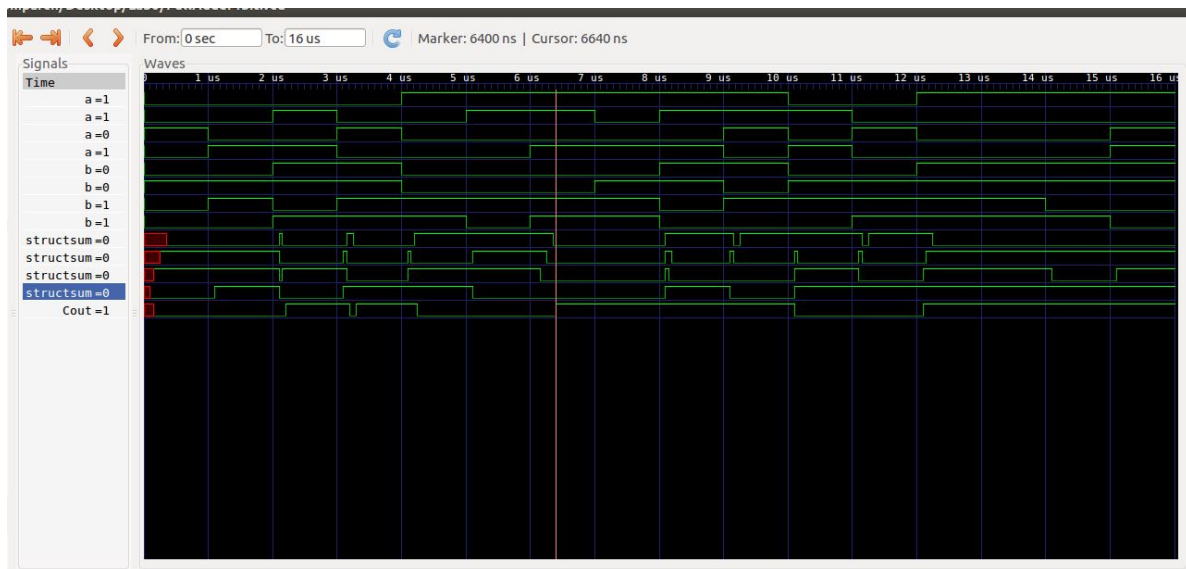


Figure 1: The two inputs ($a[3:0]$ and $b[3:0]$) with the corresponding sum ($sum[3:0]$). Because of delays in the gates used to calculate the sum and carryouts, the final values also experiences delays, as we explained in class. This can be seen most noticeably in the final “Cout” value, which is the most significant bit. This experiences delays of up to 400ns, although this would be a delay of 8, while from the activity we did in class on Tuesday, we believe this value should be 9. Similarly, we expect the maximum delay in the sum to be 10, while the most we saw was 7. We are not sure if we improperly structured some aspect of our verilog code to cause this. Tiny blips occur in the “structsum” values when not all gates have appropriately changed yet. These problems exist because the sum and carryout values for each place, starting from the least significant, rely on the value calculated from the last sum and carryout. This explains why the “structsum” values become more delayed as you go up in the image (going from least to most significant bits).

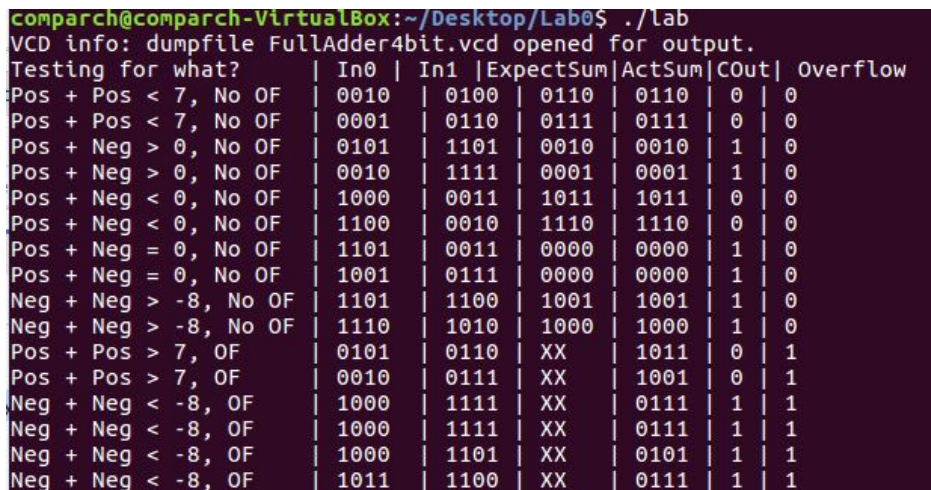
3 Testing with Verilog

We decided to test a small subset of the possible combinations of a and b inputs since otherwise there are 256 different possible combinations. We considered different types of scenarios and assumed that if the

simulation worked for that particular case it would work for all similar cases. For each case listed in table

A =	B =	Sum	Overflow
Positive	Positive	$0 < \text{sum} \leq 7$	No
Positive	Negative	$0 < \text{sum} \leq 7$	No
Positive	Negative	$-8 \leq \text{sum} < 0$	No
Negative	Negative	$-8 \leq \text{sum} < 0$	No
Positive	Negative	$\text{sum} = 0$	No
Positive	Positive	$\text{sum} > 7$	Yes
Negative	Negative	$\text{sum} < -8$	Yes

Table 1: Different scenarios we imagined the adder having to deal with. Carryout can be either 1 or 0 for each of these scenarios. Note that these are NOT the entirety of the tests we ran, but rather an example of how we thought about choosing test cases that would encompass a variety of inputs and outputs. We ran a few versions of each of these possibilities for better coverage of possible sums.



```

comparch@comparch-VirtualBox:~/Desktop/Lab0$ ./lab
VCD info: dumpfile FullAdder4bit.vcd opened for output.
Testing for what? | In0 | In1 | ExpectSum | ActSum | COut | Overflow
Pos + Pos < 7, No OF | 0010 | 0100 | 0110 | 0110 | 0 | 0
Pos + Pos < 7, No OF | 0001 | 0110 | 0111 | 0111 | 0 | 0
Pos + Neg > 0, No OF | 0101 | 1101 | 0010 | 0010 | 1 | 0
Pos + Neg > 0, No OF | 0010 | 1111 | 0001 | 0001 | 1 | 0
Pos + Neg < 0, No OF | 1000 | 0011 | 1011 | 1011 | 0 | 0
Pos + Neg < 0, No OF | 1100 | 0010 | 1110 | 1110 | 0 | 0
Pos + Neg = 0, No OF | 1101 | 0011 | 0000 | 0000 | 1 | 0
Pos + Neg = 0, No OF | 1001 | 0111 | 0000 | 0000 | 1 | 0
Neg + Neg > -8, No OF | 1101 | 1100 | 1001 | 1001 | 1 | 0
Neg + Neg > -8, No OF | 1110 | 1010 | 1000 | 1000 | 1 | 0
Pos + Pos > 7, OF | 0101 | 0110 | XX | 1011 | 0 | 1
Pos + Pos > 7, OF | 0010 | 0111 | XX | 1001 | 0 | 1
Neg + Neg < -8, OF | 1000 | 1111 | XX | 0111 | 1 | 1
Neg + Neg < -8, OF | 1000 | 1111 | XX | 0111 | 1 | 1
Neg + Neg < -8, OF | 1000 | 1101 | XX | 0101 | 1 | 1
Neg + Neg < -8, OF | 1011 | 1100 | XX | 0111 | 1 | 1

```

Figure 2: A screenshot of our testbench results. We list expected values that we calculated by hand for those scenarios with no overflow. However, we do not include expected values for those with overflow, since we anticipate there to be overflow. For each scenario we verified our results by hand with a 2s complement circle at hand to ensure we were entering correct values.

Strategy

We chose the test cases we did so that all possibilities for types of outputs were covered: negative sum, positive sum, no overflow, with overflow, no carry out, and with carry out. Since all of these worked, we believe that all types of possible inputs will yield the correct result.

Failures

We only had one failed test case at the beginning, and it was due to an error on our part rather than an error in the test case. After fixing the incorrect logic gate, the test bench yielded the results we expected.

While our final code does yield results as we expect, these results are not necessarily the “correct” answer. In the case of an overflow, the calculated result will NOT be the actual mathematical answer. However, since we indicate when an overflow has occurred, we are still able to evaluate whether we are testing correctly.

When testing on the FPGA, we initially got a couple failures/ inaccurate results even though our test bench indicated that our code was correct. After a few more tests, we found that our button 1 was not perfect, and that b was not always updating. We remedied this by holding down the button and pressing it twice, which then yielded the results we expected.

4 Testing on FPGA

To test on the FPGA, we used the provided Lab0 wrapper and uploaded our verilog file. As per the comments in the wrapper, our steps to evaluate test cases are as follows:

- Flip the switches to indicate the first value for the adder and press button 0
- Flip the switches to indicate the second value for the adder and press button 1
- Press button 2 to view the calculated sum in LEDs above the switches
- Press button 3 to view the carryout and overflow from the calculation. Carryout is displayed LED 0 (1 if there is carryout, 0 if not) and overflow is displayed on LED 1 (1 if there is overflow, 0 if there is not)

We tested the test cases in the previous table on the FPGA to validate that it was working in the same way that our test bench was. We validated each test case, and since each one matched we deduced that the FPGA was accurately reflecting our Verilog code, which was accurately portraying a 4 bit full adder. On the next two pages you can see images from two of our test rounds with the FPGA.

Examples of our FPGA operation: $0010 + 0100$

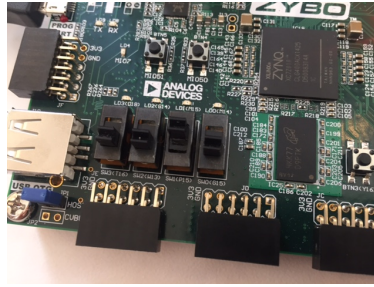


Figure 3: Setting value $a = 0010$

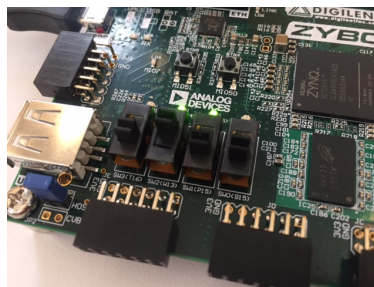


Figure 4: Setting value $b = 0100$

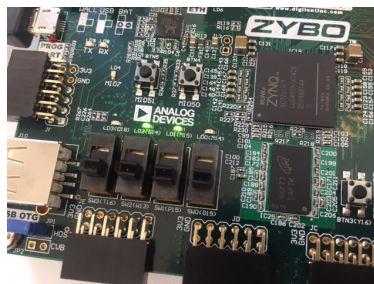


Figure 5: Viewing sum of $0010 + 0100 = 0110$

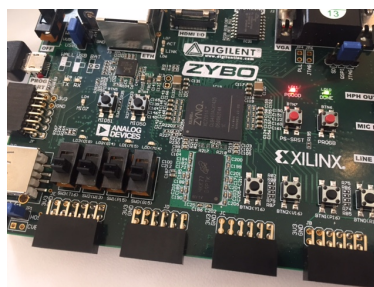


Figure 6: Viewing carryout = 0 and overflow = 0 of $0010 + 0100$

Examples of our FPGA operation: $1000 + 1101$

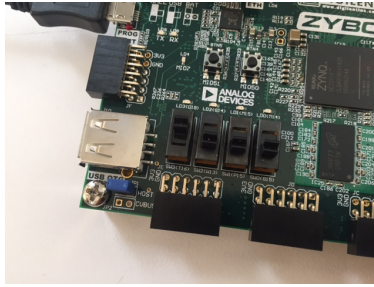


Figure 7: Setting value $a = 1000$

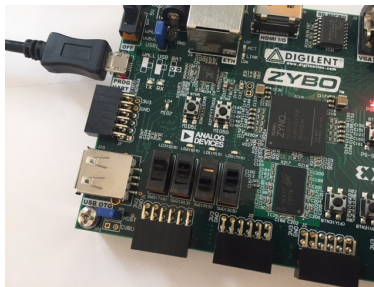


Figure 8: Setting value $b = 1101$



Figure 9: Viewing sum of $1000 + 1101 = 0101$



Figure 10: Viewing carryout = 1 and overflow = 1 of $1000 + 1101$

Vivado Synthesized Design

The report shows that we used a significant amount of inputs and outputs, but otherwise used very few of our available resources (figure 13). This can also be seen in our power utilization figure (figure 11), as the largest amount of power was consumed by I/O pins.

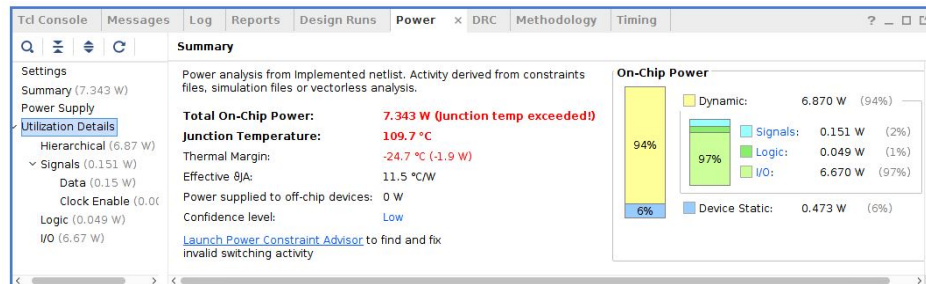


Figure 11: Data from the Synthesized Design report.

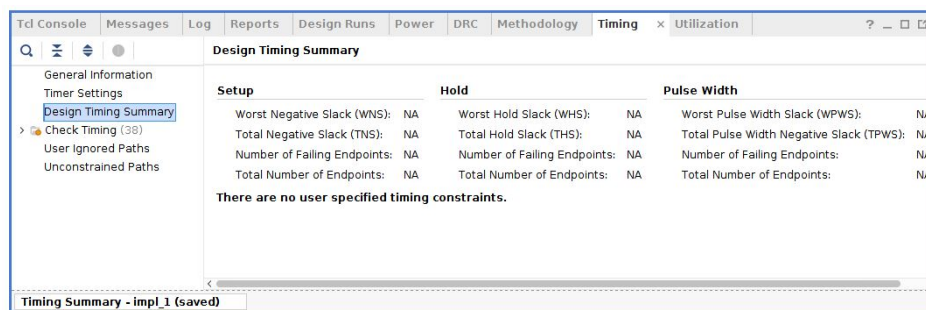


Figure 12: Data from the Synthesized Design report.

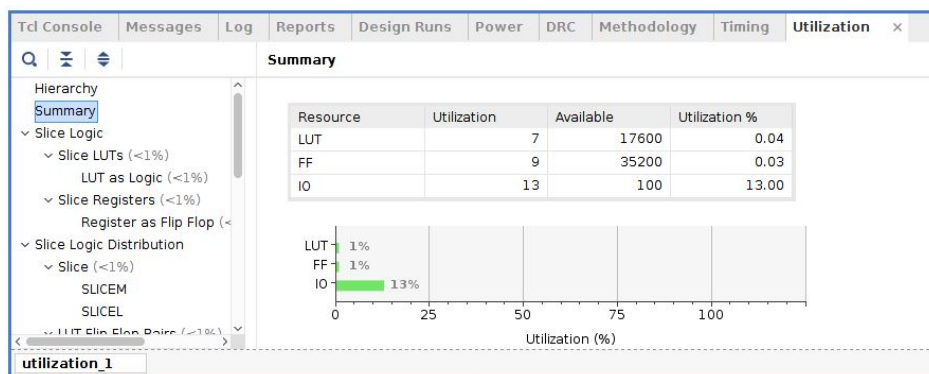


Figure 13: Data from the Synthesized Design report.