Computer Architecture Lab 0: Four Bit Full Adder

Daniel Connolly, Josh Deng, and Noah Rivkin

September 27, 2018

1 Simulation Waveforms

We ran test cases on our Verilog code prior to synthesizing it and uploading it to the FPGA. We used GTKWaveform to view the signals over time. In figure 1 we show the waveform output from eight of our test cases.

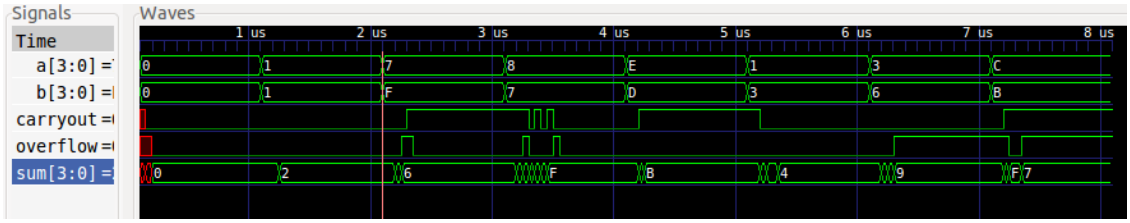

Figure 1: The waveform output from the simulated 4 bit adder

In the third test we have transient behavior for a longer duration than any of the other tests, as shown in figure 2. The test which generated the behavior was $7_{10} + -1_{10}$, or $0111_2 + -1111_2$. We believe that the cause of the transient behavior was the numerous carryover bits. The carryout bit from a 1 bit adder would only change after a delay. Since the chained adders rely on one another's carryover bits, transient behavior persisted until the carryover bit from each adder had moved through all the later adders in the chain. This resulted in a longer delay than the other test cases.

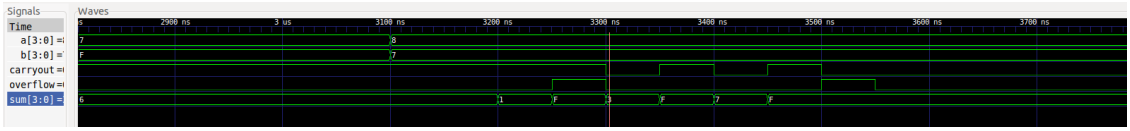

Figure 2: The greatest delay in the waveform, during test case 3. The transient behavior lasted approximately 500ns. In the test shown we were testing $7_{10} + -1_{10}$.

2 Test Case Strategy

Table 1 below displays the 16 test cases we chose. Elaborations on our test case selections is given below. Each row in the table correspond to an equation listed below. Their order matches.

1. The "zero" case:

$$0_{10} + 0_{10} = 0_{10} \tag{1}$$

2. The "all one" case:
$$-1_{10} + -1_{10} = -2_{10} \tag{2}$$

3. Adding a positive integer to another integer without overflowing:
$$1_{10} + 1_{10} = 2_{10} \tag{3}$$
$$1_{10} + 3_{10} = 4_{10} \tag{4}$$

4. Adding a positive integer to a negative integer to get a positive integer without overflowing:
$$7_{10} + -1_{10} = 6_{10} \tag{5}$$
$$4_{10} + -2_{10} = 2_{10} \tag{6}$$

5. Adding a positive integer to a negative integer to get a negative integer without overflowing:
$$7_{10} + -8_{10} = -1_{10} \tag{7}$$
$$5_{10} + -7_{10} = -2_{10} \tag{8}$$

6. Adding a negative integer to a negative integer to get a negative integer without overflowing:
$$-2_{10} + -3_{10} = -5_{10} \tag{9}$$

7. Adding a positive integer to a negative integer to get zero:
$$5_{10} + -5_{10} = 0_{10} \tag{10}$$

8. Adding zero to a positive integer to get back the same positive integer:
$$6_{10} + 0_{10} = 6_{10} \tag{11}$$

9. Adding zero to a negative integer to get back the same negative integer:
$$-5_{10} + 0_{10} = -5_{10} \tag{12}$$

10. Adding a positive integer to another positive integer to get a positive integer with overflowing. A 4-bit two complement can only cover -8_{10} to 7_{10}:
$$3_{10} + 6_{10} = 9_{10} \tag{13}$$
$$7_{10} + 7_{10} = 14_{10} \tag{14}$$

11. Adding a negative integer to another negative to get a negative integer with overflowing. A 4-bit two complement can only cover -8_{10} to 7_{10}:
$$-4_{10} + -5_{10} = -9_{10} \tag{15}$$
$$-8_{10} + -8_{10} = -16_{10} \tag{16}$$

As shown in all the cases above, we were able to obtain the correct outputs in binary fashion, including *Carryout*, *Sum*, and *Overflow*. The test cases we choose covered all the modes of operations the adder is expected to have. As a result of the fact that the adder was able to function correctly in all these cases, we are confident that we can avoid listing all 256 cases to prove functionality by exhaustion.

Table 1: Four-bit Adder Test Cases

a	b	Cout	Sum	Overflow	Expected Output (Base 10)
0000	0000	0	0000	0	0
1111	1111	1	1110	0	-2
0001	0001	0	0010	0	2
0001	0011	0	0100	0	4
0111	1111	1	0110	0	6
0100	1110	1	0010	0	2
1000	0111	0	1111	0	-1
0101	1001	0	1110	0	-2
1110	1101	1	1011	0	-5
0101	1011	1	0000	0	0
0110	0000	0	0110	0	6
1011	0000	0	1011	0	-5
Overflow Test:					
0011	0110	0	1001	1	9/Overflow
0111	0111	0	1110	1	14/Overflow
1100	1011	1	0111	1	-9/Overflow
1000	1000	1	1000	1	-16/Overflow

3 Summary of Testing on FPGA board

On the Zybo FPGA board, we tested all of the cases shown in Table 1. In order to test the cases, we first loaded our numbers in binary onto the board by changing the position of switches zero through three, at the bottom left of each picture in Figure 3. To load the first number, we pressed button zero after positioning the switches; to load the second, we pressed button one. At this point, we pressed button two to display the sum on the LEDs, which are green and next to the switches in Figure 3, and button three to display the carryout and overflow. After running through all of our test cases, the LEDs seemed to be performing as we expected, confirming that we had programmed the board correctly.

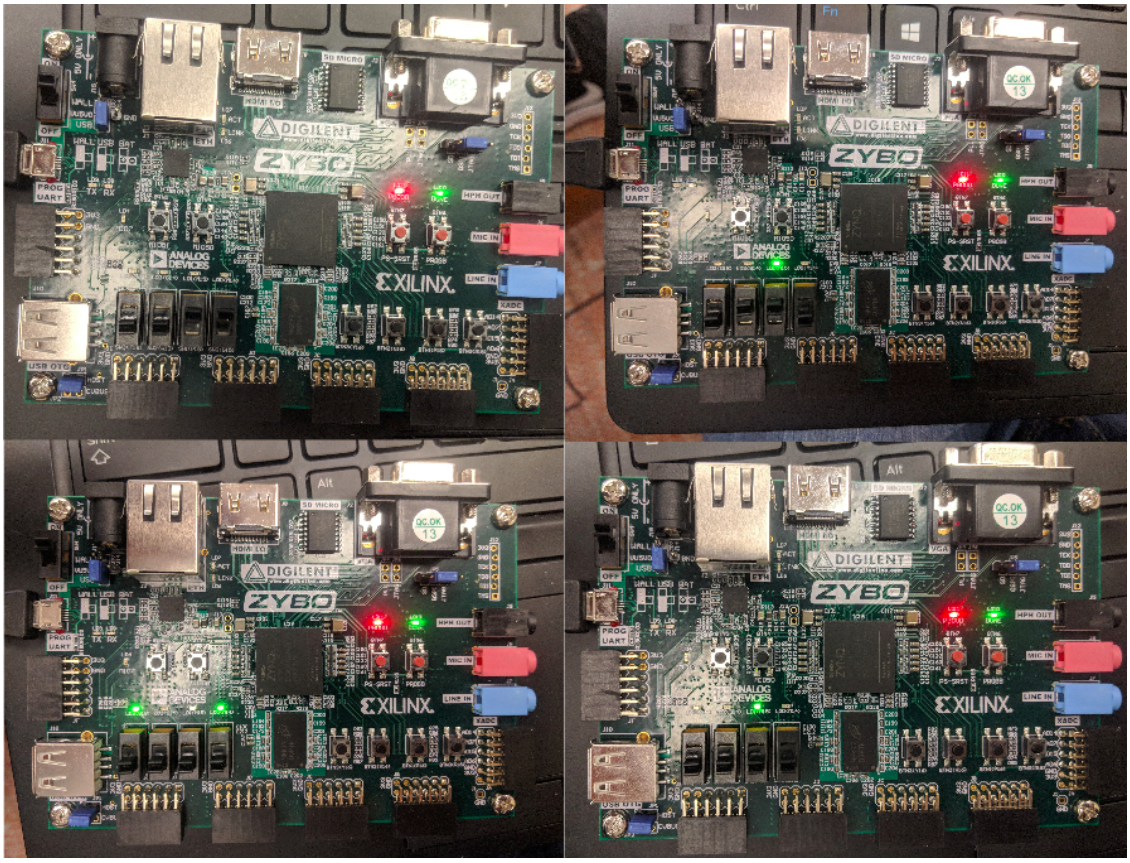

Figure 3: (Top Left) We pressed button 0 to load b0011. (Top Right) We pressed button 1 to load b0110. (Bottom Left) We pressed button 3, which displayed the sum, b1001. (Bottom Right) We pressed button 4, which displayed an overflow of 1 on LED 1 and a carryout of 0 on LED 0.

4 Summary Statistics

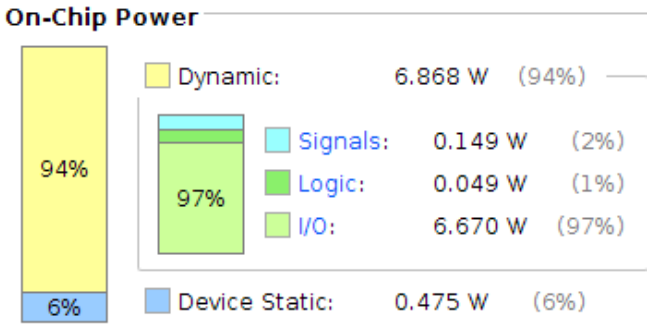

Figure 4: A summary of the FPGA power with our 4-bit adder implementation. A single four bit adder is quite a simple design so that we can see that the signal lines and logic units take relatively small percentage of the power (only 3% with the two combined). Most of the power is used to drive the I/O (Input/Output) which includes, but is not limited to, LEDs, switches, and buttons.

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	inf	Worst Hold Slack (WHS):	inf	Worst Pulse Width Slack (WPWS):	NA
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	NA
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	NA
Total Number of Endpoints:	21	Total Number of Endpoints:	21	Total Number of Endpoints:	NA

There are no user specified timing constraints.

Figure 5: The timing information for the 4-bit adder. The majority of the statistics available relate to clock information. Because the circuit uses combinational logic and does not use a clock there is little useful information available here.

Name 1	Slice LUTs (17600)	Slice Registers (35200)	Slice (4400)	LUT as Logic (17600)	LUT Flip Flop Pairs (17600)	Bonded IOB (100)	BUFGCTRL (32)
N lab0_wrapper	7	9	4	7	1	13	1
opA_mem (dff)	5	4	4	5	0	0	0
opB_mem (dff_0)	1	4	1	1	0	0	0
src_sel (jkff1)	1	1	1	1	1	0	0

Figure 6: A summary of the resources use on our FPGA is shown above. Our FPGA utilized seven look-up tables. If we were to utilize this same style of ripple-carry adder to add a greater number of bits, the number of look-up tables the FPGA would employ would scale linearly with the number of bits.

5 Test Case Failures

During the construction and the testing of our four-bit adder, we did not encounter any cases where the adder gives contradictory results. After an initial discussion, we very quickly landed on a ripple-carry style adder where we chained one bit adders together. As a result, we were able to complete the design and test bench in Verilog relatively quickly and achieved the output we expected in every case, before implementing the code on the Zybo, again without any errors. ***drops mic***