Computer Architecture Lab 0

Zarin Bhuiyan, Bonnie Ishiguro, David Zhu

September 25, 2016

1 Stabilizing Waveform

After writing our 4-bit full adder in Verilog, we ran the code in GTKWave to get a simulation of its waveform.

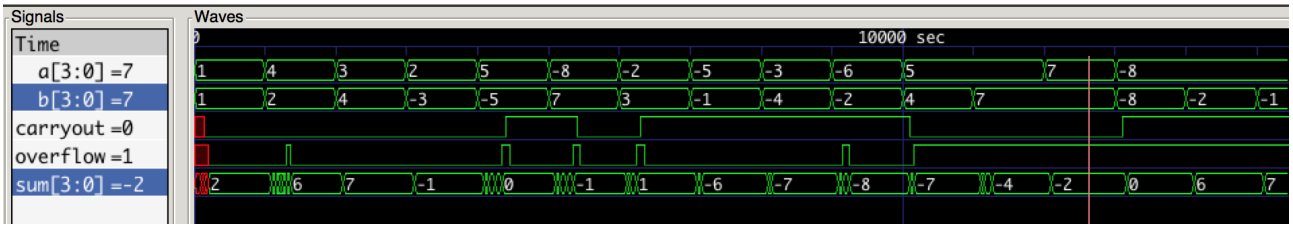

Figure 1: This is our GTKWave waveform which simulates our Verilog code.

The worst case delay shown in our simulation was the addition of $5+-5$. Checking the difference in marker locations showed that the delay was approximately 350 seconds. Each of our gates had a 50 second delay, so this means this addition went through 7 gates. Theoretically, the worst possible delay should be 400 seconds or 8 propagate delays. However, because the interface of our 4 bit adder does not have a carry in for the least significant bit we could not get a state where we had 1 additional propagation.

This is because our 1-bit full adder is built with an AND gate for the CarryIn. By not exposing the CarryIn, the AND gate was already stable, and thus the SUM signal for the first adder only when through 1 propagation delayed gate, reducing our theoretical 400s to 350s.

2 Test Case Strategy

We chose the sixteen test cases to test both our Verilog code and our full adder on the FPGA. They included sums of positive addends with and without overflow, of negative addends with and without overflow, and of a negative and a positive addend with and without overflow. We also included cases where the addends sum to zero. The unique configurations we chose are shown in the table below.

A	B	Sum	CarryOut	Overflow	ESum	ECarryout	EOverflow
0001	0001	0010	0	0	0010	0	0
0100	0010	0110	0	0	0110	0	0
0011	0100	0111	0	0	0111	0	0
0010	1101	1111	0	0	1111	0	0
0101	1011	0000	1	0	0000	1	0

1000	0111	1111	0	0	1111	0	0
1110	0011	0001	1	0	0001	1	0
1011	1111	1010	1	0	1010	1	0
1101	1100	1001	1	0	1001	1	0
1010	1110	1000	1	0	1000	1	0
0101	0100	1001	0	1	1001	0	1
0101	0111	1100	0	1	1100	0	1
0111	0111	1110	0	1	1110	0	1
1000	1000	0000	1	1	0000	1	1
1000	1110	0110	1	1	0110	1	1
1000	1111	0111	1	1	0111	1	1

3 List of Key Failures

Our first key failure was our formula for calculating overflow. We first assumed that overflow could be calculated by taking the XOR of the final carryout and the most significant bit of the final sum. After running our test bench on our four-bit adder, however, we discovered that this fails for the cases where either the carryout bit is 0 and the most significant sum bit is 1, or where the carryout bit is 1 and the most significant sum bit is 1.

We then assumed that the overflow could be calculated with the below formula, based on the selection of test cases that we had implemented at the time.

$$overflow = carryout \cdot \overline{s_3} \tag{1}$$

However, after further test cases, we discovered that overflow is present when the final two carryout values are not equal. We then revised our overflow formula to be the XOR of the final carryout and the carryout of the most significant bit column, which passed our sixteen test cases.

$$overflow = carryout \oplus carryout_2 \tag{2}$$

Another thing we struggled with was finding a test case the reproduces the worst case propagation delay of 400s. The issue was mathematically mis-calculated, since we used a 4 full-adders instead of 3 full-adders and 1 half-adder. The worst case for four full-adders of our implementation was 8 timeunits for the sum. However, in reality the first adder would never receive a carry-in, and we could never reproduce our worst delay case in reality because of this.

Looking further first full-adder, we figured out the root of this issue updated our propagation delay calculations to match. Our visual representation from Verilog also demonstrates this.

4 Summary of Testing on FPGA

Tests on the FPGA match the 16 tests written for our Verilog testbench. After much trial and tribulation, we were finally able to load the FPGA with our most updated code. The physical testing of the 4-bit adder matches our Verilog testing at the end.

Here is an example where we add 0b1011 and 0b1111.

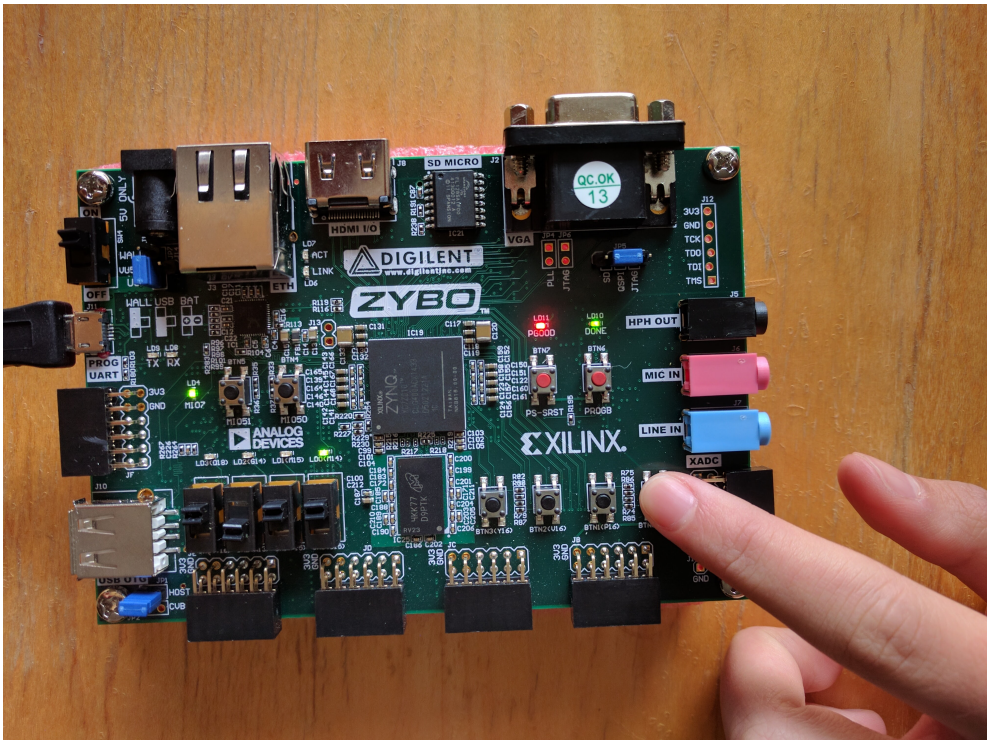

Figure 2: Choose A, the first addend, by arranging the switches to represent the number in binary and then press b0, the first button.

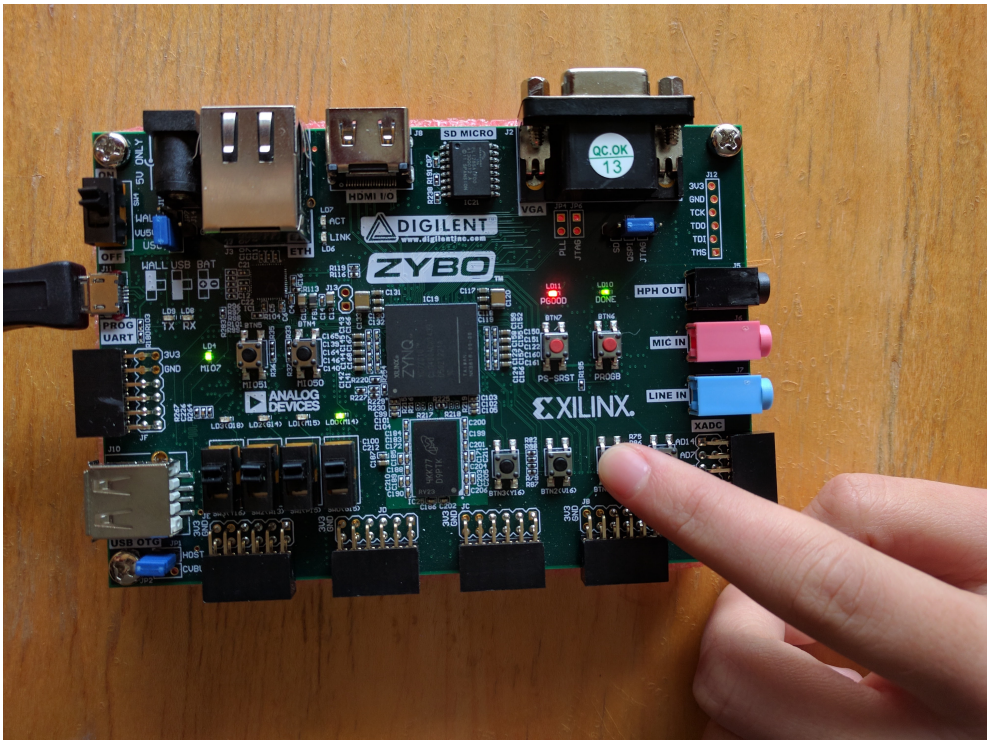

Figure 3: Choose B, the second addend, and press b1.

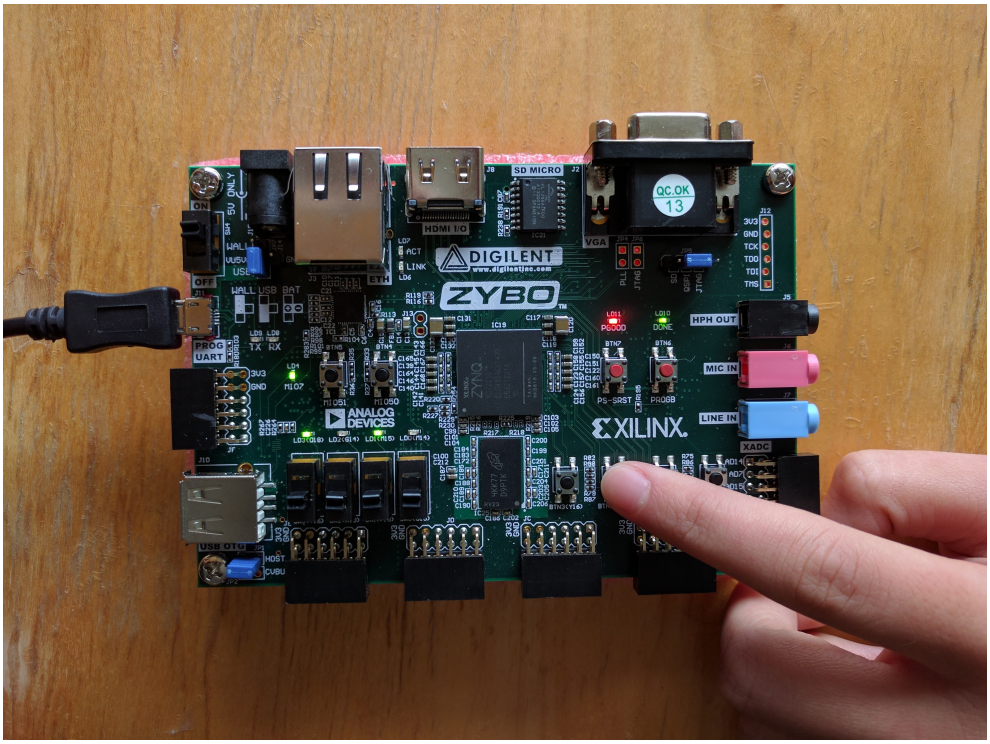

Figure 4: Press b2 to display the calculated sum of A and B (0b1010), represented by the LEDs above the switches.

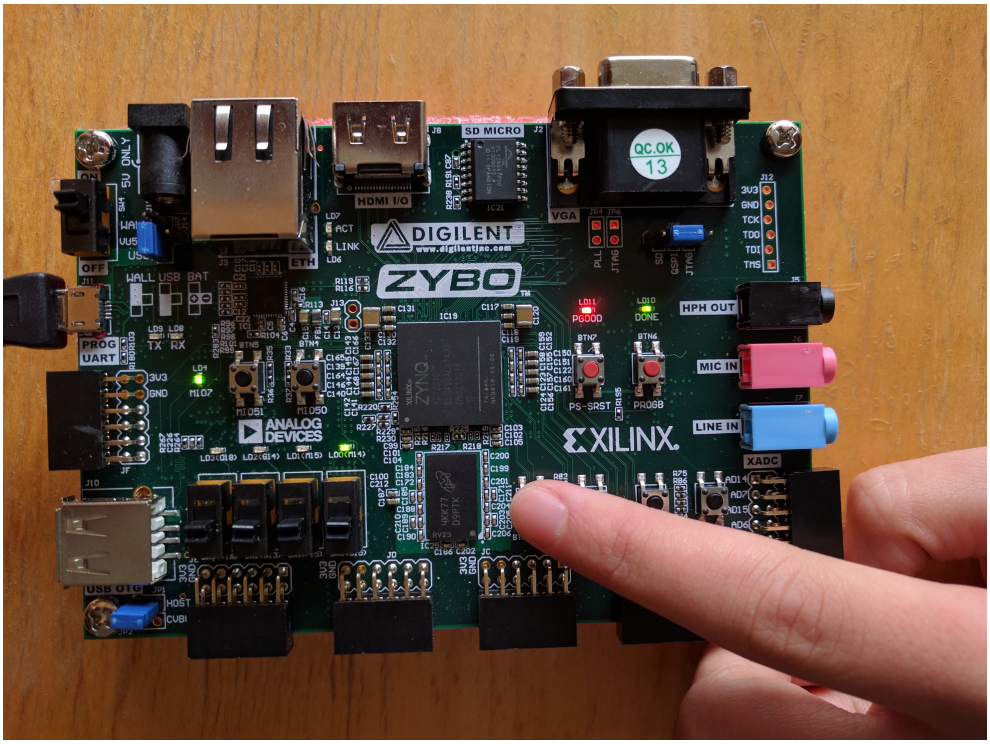

Figure 5: Press b3 to display the carryout (1) with the right-most LED and the overflow (0) with the second LED to the right.

5 Summary Statistics

Delay Propagation

The 4-bit adder is a chain of our 1-bit full adders that follow the standard design, as opposed to a lookup table. Based on that, we know that:

$$Delay_{sum} = 2N = 2(4) = 8 time units$$

$$Delay_{carryout} = 2N + 1 = 9 time units$$

$$Delay_{overflow} = Delay_{carryout} + 1 = 10 time units$$

With corrections for the stable un-exposed carry-in,

$$Delay_{sum} = 7$$

$$Delay_{carryout} = 8$$

$$Delay_{overflow} = 9$$

Resources Used

The sub-modules that we used are LUT (Look Up Tables), FF (Flip Flops), IO, and BUFG (Clock Buffer). The 8 LUTs store the 4 bits for the addend A and the 4 bits for the addend B, and the 13 IOs correspond to the LEDs, switches, buttons, and clock.